NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

AN OBJECT-ORIENTED DISCRETE-EVENT SIMULATION OF LOGISTICS (MODELING FOCUSED LOGISTICS)

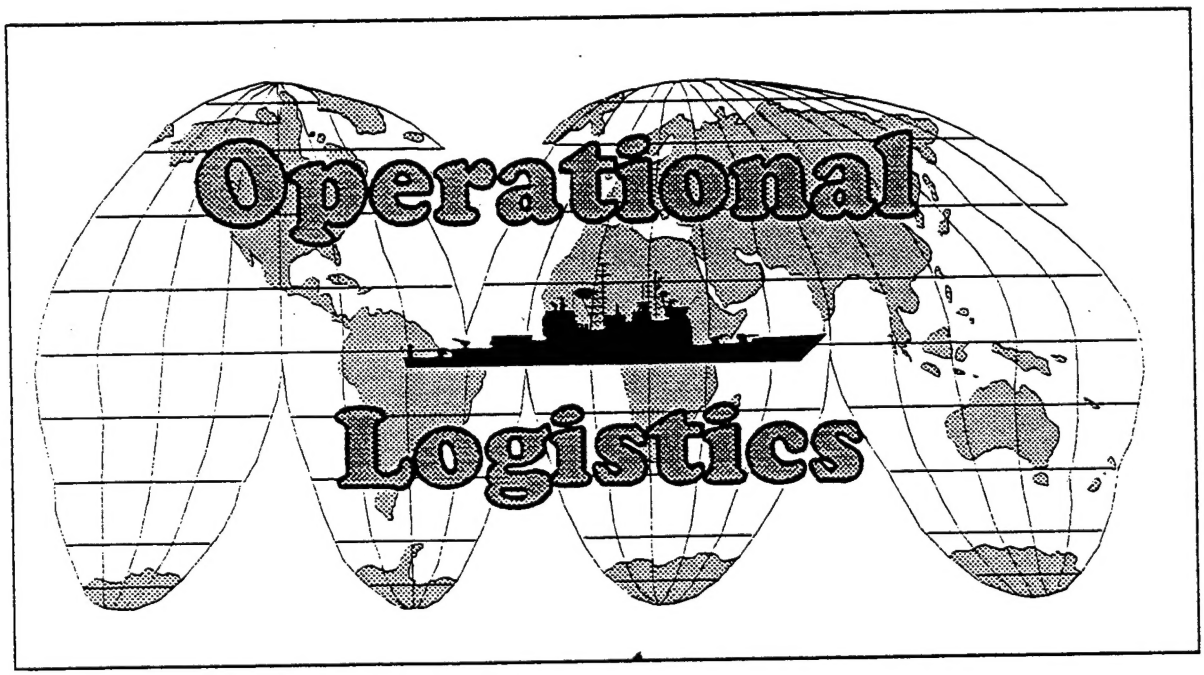
by

John L. Ruck

September 1998

Thesis Advisor: Arnold H. Buss
Second Reader: Gordon Bradley

Approved for public release; distribution is unlimited.


Operational
Logistics

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September, 1998	Master's Thesis

4. TITLE AND SUBTITLE
AN OBJECT-ORIENTED DISCRETE-EVENT SIMULATION OF LOGISTICS (MODELING FOCUSED LOGISTICS)

5. FUNDING NUMBERS
DSAM80008

6. AUTHOR(S)
Ruck, John L.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Office of the Secretary of Defense, Program Analysis and Evaluation, Pentagon room 2E314, Washington, D.C. 20301-1800

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

Joint Vision 2010 puts forth four operational concepts describing how U.S. forces will conduct combat in the future. One of these concepts is Focused Logistics, which Joint Vision 2010 defines as "the precise application of logistics." In order to study the effects of Focused Logistics, a flexible method of simulating possible logistics systems is needed. The Flexible Experimental Logistics Simulator (FLEXLOGS) is an object-oriented, discrete-event simulation that is designed to be used to evaluate proposed future logistics strategies. First, the author develops a model capable of simulating any proposed logistics scheme with minimal modification to the software. Second, the thesis discusses the design and use of the model. Finally, the model is used to explore the shape of curves defined by the probability of combat victory verses "logistical footprint size" and "premium transportation availability." The model implements the draft Logistical Conceptual Object Model being developed as part of the *Focused Logistics Study* by the Office of the Secretary of Defense, Program Analysis and Evaluation, Simulation Analysis Center.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Focused Logistics, FLEXLOGS,.Discrete-Event Simulation	110
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

Approved for public release; distribution is unlimited

AN OBJECT-ORIENTED DISCRETE-EVENT SIMULATION OF LOGISTICS (MODELING FOCUSED LOGISTICS)

John L. Ruck
Lieutenant Commander, United States Navy
B.S.E., Tulane University, 1982
M.S., Central Michigan University, 1995

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
September 1998

Author: _____
John L. Ruck

Approved by: _____
Arnold H. Buss, Thesis Advisor

_____
Gordon Bradley, Second Reader

_____
Richard Rosenthal, Chairman
Department of Operations Research

ABSTRACT

Joint Vision 2010 puts forth four operational concepts describing how U.S. forces will conduct combat in the future. One of these concepts is Focused Logistics, which Joint Vision 2010 defines as "the precise application of logistics." In order to study the effects of Focused Logistics, a flexible method of simulating possible logistics systems is needed. The Flexible Experimental Logistics Simulator (FLEXLOGS) is an object-oriented, discrete-event simulation that is designed to be used to evaluate proposed future logistics strategies. First, the author develops a model capable of simulating any proposed logistics scheme with minimal modification to the software. Second, the thesis discusses the design and use of the model. Finally, the model is used to explore the shape of curves defined by the probability of combat victory verses "logistical footprint size" and "premium transportation availability." The model implements the draft Logistical Conceptual Object Model being developed as part of the *Focused Logistics Study* by the Office of the Secretary of Defense, Program Analysis and Evaluation, Simulation Analysis Center.

THESIS DISCLAIMER

The reader is cautioned that computer programs developed in this research may not have been exercised for all cases of interest. While every effort has been made, within the time available, to ensure that the programs are free from computational and logic errors, they cannot be considered validated. Any application of these programs without additional verification is at the risk of the user

TABLE OF CONTENTS

TABLE OF FIGURES

EXECUTIVE SUMMARY

Joint Vision 2010 (JV2010) is the Chairman of the Joint Chiefs' plan for how military operations will be conducted in the future. One of the operational concepts put forth in JV2010 is Focused Logistics, defined in JV2010 as the "precise application of logistics." To fully define what Focused Logistics will mean to US forces, the Office of the Secretary of Defense (OSD), Program Analysis and Evaluation (PA&E) branch is conducting the Focused Logistics Study. As part of the Focused Logistics Study, the Simulation Analysis Center (SAC) is engaged in the Modeling Experiment in Focused Logistics to explore how to better model logistics in the future. As a part of that modeling experiment, this thesis develops a flexible object-oriented logistics simulation based on the draft Conceptual Object Model of Military Logistics being developed as part of the Focused Logistics Study. Using the conceptual object model as a starting point, the author developed the "Flexible Experimental Logistics Simulator (FLEXLOGS)." FLEXLOGS is an object-oriented discrete-event simulation implemented in the Java programming language. The model treats the logistics system as a transportation network, with the requisite rules for ordering and shipping material. The model consists of objects representing the units involved in combat and logistics support, the transportation assets, the material and information in the logistics system, and the network of transportation links.

One of the key design goals of the model is flexibility so that the model can be easily adapted to study a wide range of possible future logistics systems. Two methods were employed to meet the goal of flexibility. First, as much flexibility as possible is built into the design to allow the nature of the logistics network and rules for ordering supplies to be controlled by an input data control file. Second, the software employs a modular design so that modifications to logic contained in the computer code can easily be made.

The model is used to address the question of how combat effectiveness is affected by the amount of supply in theater and the amount of available premium transportation.

The relationship of these two factors is shown as a three dimensional surface. To explore sensitivity of the MOE to changes in the logistics system, the rules for re-supply among units in theater are relaxed. The model demonstrates that this relaxation does not produce the expected improvement in combat effectiveness. The model runs were examined more closely to determine that the limiting factor in this scenario is premium strategic lift, so a relaxation of theater ordering rules would have no effect. Next, the model was modified so that strategic lift was not a limiting factor. Running the model under these conditions revealed that relaxing the ordering rules does not necessarily improve the combat effectiveness and, in fact, in this case made the outcome of the combat less predictable.

This thesis demonstrates that a flexible simulation can be developed from a conceptual model and used to explore the nature of the system under study. FLEXLOGS provides a good tool with which to explore the possible effects of future logistics systems.

ACKNOWLEDGEMENTS

The author would like to thank the Office of the Secretary of Defense, Program Analysis and Evaluation branch for funding the experience tour that made this thesis possible. The author would like to specifically thank Mr. Jim Johnson, LtCol Charles Mugno, USMC, and Dr. Kevin Seager for their help and guidance in developing the model. The author would also thank Professors Arnold Buss and Gordon Bradley for their patience and help during the writing of this thesis.

I. INTRODUCTION

The United States military is current undergoing a "revolution in military affairs" that will cause a drastic change in the way that all services conduct operations [Owens, 1995]. A number of factors have brought about these changes. First, the fall of the Soviet Union has changed the nature of the potential enemy from a single large advisory to a large number of possible foes. We can no longer plan on the war occurring in a specific place under specific conditions. Our forces must therefore be more mobile and flexible than they were during the Cold War. Second, the reduction of the threat from an easily identifiable enemy has caused the resources available for defense to shrink over the years, requiring the military to stretch tax dollars further. Therefore, today's military must not only be flexible and mobile, but also cost-effective.

Logistics is one of the key areas that must change in order to enable this revolution. In the past, the U.S. armed forces solved logistics problems with brute force in which large quantities of material were stockpiled in theater for planned operations, with even larger amounts of materials being shipped into the theater at the onset of hostilities. This method of supplying troops runs counter to the contemporary requirements for logistics to be flexible, mobile, and efficient. Furthermore, in future conflicts the large stockpiles of materials could be a liability rather than an asset.

In order to conduct operations with fewer supplies in theater, the same emphasis must now be placed on logistics that is currently placed on developing and analyzing the war plan. This increased emphasis on analysis will require that logistics simulations be

brought up to the capability of combat simulation. Planners and analysts need a robust, flexible logistics simulation.

A. BACKGROUND

Joint Vision 2010 (JV2010) is the Chairman of the Joint Chiefs of Staff's vision of how the United States military will operate in the 21st century. The Chairman lays out four operational concepts required for the successful conduct of warfare in the future. One of these concepts is "Focused Logistics," defined in JV2010 as:

> The fusion of logistics information and transportation technologies for rapid crisis response; deployment and sustainment; the ability to track and shift units, equipment, and supplies even while en route, and delivery of tailored logistics packages and sustainment directly to the warfighter [Joint Vision 2010, p24].

To further define this new concept of logistics, the Joint Chiefs published *Focused Logistics*. One of the key features required to implement Focused Logistics is the reduction of the amount of materials present in theater. This reduction helps units to deploy more rapidly due to reduced transportation requirements. Since *Focused Logistics* discusses high level concepts, more research is needed to define the technologies and procedures that will be Focused Logistics.

To further refine the concept of Focused Logistics, the Office of the Secretary of Defense, Program Analysis and Evaluation (OSD, PA&E) has undertaken the *Focused Logistics Study*. Through this study, OSD will lay out how the concept of Focused

Logistics should be implemented. In order to evaluate possible future logistics concepts, the analysts will require some type of modeling tool [SAC briefing, 1997].

The *Experiment in Modeling Focused Logistics* is an effort to determine how future logistics concepts can be modeled. The experiment has three main objectives. First, to provide a tool that will be able to quantify the benefits and risks of Focused Logistics. Second, to assess the suitability of currently available analysis tools for studying future logistics concepts. Third, to develop recommendations for the logistics model in JWARS. The second of these objectives will provide a foundation to accomplish the other two [SAC briefing, 1997].

The modeling experiment will attempt to answer three questions about logistics modeling. First, how do we currently model logistics? Second, how will we model today's logistics in the future? Finally, how will we model Focused Logistics?

Some preliminary work has already been accomplished in searching existing models. The modeling experiment started with TACWAR 5.0, since it is the combat model most widely used in OSD, PA&E studies. Although TACWAR does include a simple logistics model, it may not be suitable for accurately modeling current and future logistics. Most of the currently available logistics models contain embedded logic that "hard-code" the logistics rules, and consequently lack the flexibility necessary for exploring innovative logistics concepts. One of the areas that is under investigation is the creation of a federation of models (including both combat and logistics models) using the High Level Architecture (HLA) [Seager, 1998].

As an adjunct to the modeling experiment, the director of OSD, PA&E, Simulation and Analysis Center (SAC) has requested the author of this thesis to construct an object-oriented logistics simulation based on the *Conceptual Model of Military Logistics* (COM), currently under development as part of the Focused Logistics Study [Draft COM, 1997]. The director also requested that the author conduct an initial investigation to discover the effect of reducing stockpile size on combat effectiveness.

B. PREVIOUS RESEARCH

The literature on the specific subject of simulating the effects of logistics on combat is very scarce. Most of the previous research found was narrowly centered on answering a specific question and not at developing a more general method for studying a wide range of possible scenarios and logistics solutions. The paragraphs that follow discuss the highlights of some of this research.

A series of three Naval Postgraduate School (NPS) theses were written to provide a tool for a wargame referee to use to determine the effect of logistics on a war. The first of this series [Long, 1993] developed a simulation of a logistics network with the purpose of providing logistics information to the referee and participants of a theater level wargame. Long's thesis provided a good starting point for this logistics referee. Two more NPS students improved upon the original design [Halverson, 1994; Plunk, 1995] by implementing improvements noted in the original thesis. This series of theses was useful to the author for learning some general techniques for simulating the logistics network. However, this author's lack of familiarity with the modeling language chosen, MODSIM,

and the lack of comments in the code provided, hindered the direct application of methods developed in these theses to the current research. Also, these theses were focused on solving specific problems, rather that developing more generic modeling solutions. The simulations developed were not designed to be coupled directly with a combat simulation, but rather to be used by a human referee to manually transfer the effects of logistics from this model to the wargame in progress.

In a thesis written in 1996, Huffaker developed a simulation, also written in MODSIM, to demonstrate the effects of logistics on combat. His thesis focused on computing the level of supply available at a combat unit for a given scenario. It included a combat simulation and also a simple supply interdiction model. The model was specifically designed to model the logistics of ground combat and used a built-in combat and interdiction model as the source of demands.

A study of the Marine Corps' Enhanced Transportation Service Program by Atchesonn [1997], showed that using premium transportation, overnight airfreight in this case, could be a cost effective alternative to carrying higher stock levels. While the study focused on a small number of very specific scenarios, it is useful since it shows that replacing stockpiles with transportation is both feasible and cost effective.

In his 1996 thesis, Stork was one of the first to use the Java language for discrete-event object-oriented simulation. While the purpose of Stork's thesis was to study how to model sensors, it provided an excellent foundation for the use of Java as a modeling language. In the thesis he developed the first version of "Simkit", a toolkit for simulation in Java.

The use of discrete-event object-oriented simulation in Java is not limited to military modeling. As an example, Caoyuan Zhong, in his PhD dissertation [1997] at Virginia Polytechnic Institute, developed a flexible model of airport operations in order to study the effect on airport congestion of various changes in policy. Another purpose of Zhong's dissertation was to explore the use of Java in simulations. He found that airport simulations also suffered from maintainability problems caused by large, procedural languages and that this area of simulation could benefit from the use of an object-oriented language.

While these previous simulations provide a solid background and are a good source of general simulation methodology, none of them can be applied directly to study the effects of Focused Logistics.

C. OUTLINE

The remainder of this thesis will document the development and use of a flexible tool for modeling logistics. Chapter II will present an overview of the Conceptual Object Model of Military Logistics upon which FLEXLOGS is based. Chapter III is an overview of the implementation of the model, with detailed information contained in Appendix B and Appendix C. Chapter IV uses the model to explore the effect of changes in logistics on combat effectiveness. Finally, Chapter V presents the conclusions and recommendations of this thesis.

II. THE CONCEPTUAL MODEL

A. THE PURPOSE OF A CONCEPTUAL MODEL

A "Conceptual Object Model" is a high-level model of the objects present in the system under study that identifies the major entities, relationships, and business functions. A conceptual model is not intended to be a design for a system to simulate the real world. Rather, it provides an organized way for an analyst to document the workings of the system of interest and a framework for the modeler to document the system without getting lost in needless implementation details, or restrictions caused by a particular implementation method. Only objects necessary for understanding of the system need be documented, not objects that are part of the overhead necessary for development of a working simulation.

However, a conceptual model is of great use to an analyst creating a simulation of a system. The conceptual model documents the real world objects that are of interest providing a documented boundary to the system under study and provides an excellent starting point for simulation software development. The use of the conceptual model by the simulator also proves to be a synergistic benefit to the conceptual modeler. During the implementation, the modeler can provide feedback on areas of the model he feels are lacking.

B. THE CONCEPTUAL MODEL OF MILITARY LOGISTICS

The conceptual model is currently under development at the Simulation Analysis Center. A brief outline of the draft model will be presented in the following paragraphs. Although this version of the model is not yet complete, it provides valuable insight and a foundation for a prototype flexible logistics model.

The draft model consists of three parts. The first part can be thought of as the logistics system interface to the outside world, a monolithic entity that provides services to other entities. This view allows one to define the boundary of the logistics system and define the interface to the combatant entities. This interface consists of a military logistics task list. The second part of the draft model consists of two object diagrams. One of these diagrams is an entity relationship diagram, which shows how the objects in the model interact. The other diagram is a class hierarchy diagram that shows the relationships between the objects. The final part of the model is a series of "class cards" that define the attributes and methods of the objects in the model. Before the task list can be discussed, the high-level object definitions contained in Table 1 are needed.

The military logistics system is capable of the following types of tasks: Providing, Receiving, Repairing, Transporting, and Disposing of. Note that not all of the combinations of tasks are meaningful. The combinations that the draft conceptual object model defines as making sense are listed in Table 2.

Table 1 – High Level Object Definitions

Object	Definition	Example
Asset	Something used by an organization to accomplish a task	Combatant, Equipment
Supply	Something that is consumed by an asset or liability	Food, Ammunition
Infrastructure	Fixed facilities that provide a capability	Roads, Buildings
Liability	Something that the military accepts responsibility for that does not aid in the accomplishment of the mission.	POW, Non-Combatant Evacuees.

Table 2 Service/Object Matrix (An 'X' indicates a meaningful combination of tasks)

	Asset	Supply	Infrastructure	Liability
Provide	X	X	X	
Receive		X		
Transport	X	X		X
Repair	X		X	X
Dispose of	X	X		X

The next part of the conceptual object model is the entity relationship diagram, which shows how objects can interact with each other. The diagram from the draft object model is included as Appendix A.

The Asset class hierarchy diagram is shown in Figure 1 as an example. As can be seen from the diagram, the conceptual object model contains a large amount of detail. This level of detail is necessary to fully document the interactions within a complex system.

The Conceptual Model of Military Logistics is currently under development, but still provides valuable insight into the system under study. The conceptual model also provides a good starting point for developing a simulation of military logistics.

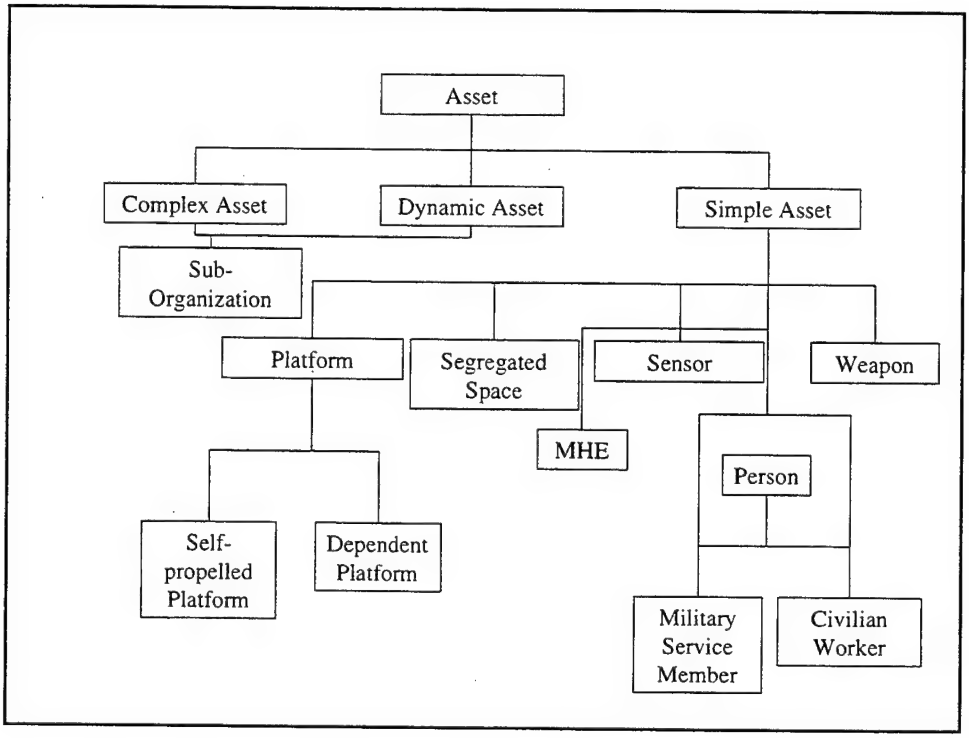


Figure 1 Asset Class Hierarchy Diagram – This diagram illustrates the large amount of detail contained in the conceptual object model.

The Conceptual Object Model, while primarily intended to provide documentation of the military logistics system at the abstract level, provides the basic framework from which to develop a working simulation. The details of the implementation of a simulation based on the Conceptual Object Model of Military Logistics are presented in the following chapter.

III. MODEL IMPLEMENTATION AND DESCRIPTION

A. OVERALL MODEL

The model for this thesis was developed using the Java programming language. The choice of language was guided by the need for an object-oriented language. The other languages that were considered were C++, Smalltalk, and MODSIM. Among the reasons for the choice of Java were the author's familiarity with the language, Java's memory management features, and the cross-platform portability of the language.

As stated earlier, the model implemented was based heavily on the Conceptual Object Model of Logistics under development by the Simulation Analysis Center. The model treats the logistics system as a transportation network, with the requisite rules for ordering and shipping material. As the main thrust of this thesis is logistics, the combat model portion of the simulation was of limited scope, and only a limited effort was devoted to its development.

The model consists of objects representing the units involved in combat and logistic support, the transportation assets, the material and information in the logistics system, and the network of transportation links. The paragraphs that follow and Appendix B will discuss the model in greater detail, with the purpose of educating the reader not only of the operation of the model, but also of the design of the model. If the model is to be used for its intended purpose of studying future logistics, then future users may need to modify some of the logic embedded in the classes.

Since the model has been designed to give a great amount of flexibility to the modeling of logistics, it was named The Flexible Experimental Logistics Simulator (FLEXLOGS).

B. STRATEGY FOR FLEXIBILITY

Since the genesis of this project was the need for a flexible logistics simulation, one of the major underlying design goals was flexibility. This flexibility was gained through a two pronged approach. The ultimate in flexibility would be to develop the software in such a way that any changes in the system under study could be implemented in the model by changing data input only. This is a worthy but unachievable goal. The second method of achieving flexibility is to design the software such that it is easily modifiable. FLEXLOGS employs both approaches to flexibility. The structure of the logistics network, stock levels, re-order points, the types and number of transportation assets, and others are all controlled through data input. In addition, its object-oriented design enables the easy addition of features. In a thesis studying the feasibility of supporting a Marine Expeditionary Unit from a sea base, Stuart [1998] was able to readily adapt the model to his problem.

When flexible implementation was too difficult or time did not allow, behavior of the model was coded into the computer program. Things that were judged by the author to most likely be changed in the course of running the model were segregated into separate Java classes for ease of modification. Examples of this include the rules for ordering supplies (i.e. who orders what from whom), the distributions of the various

random variables used in the simulation, the method of tracking and picking transportation assets needed for a shipment, and the interface to the combat model. The Java classes, which implemented this flexibility, are discussed with the rest of the simulation in the next section and Appendix B.

Another strategy was to try to make data internal to the model visible. A number of classes were developed which allow the analyst to log certain attributes of the objects to a file for further analysis.

C. OBJECT DESCRIPTIONS

The Java programming language organizes the various classes in a program into packages that separate them into functional groups. The paragraphs that follow will discuss each package at a high level. The various member classes of each package are discussed in Appendix B. Classes that provide only minor functions in the model will not be discussed. The full documentation and source code for all of the packages is too voluminous to include in this thesis, but is available from the author.[1]

1. Package Descriptions

FLEXLOGS consists of eight Java packages summarized in Table 3. The TM package contains the time master, which controls the scheduling of events in the time step model. The attribute package contains classes that are used to track and log various

---

[1] The author may be contacted via e-mail at John@Ruck.com or on the world wide web at http://www.John.Ruck.com

attributes of the entities in the model. The battle package is a simple combat simulation developed for this thesis which is used to generate the logistics demands and also used to determine the winner of the conflict for use as a measure of effectiveness. The combat package contains those classes that are the interface between the logistics simulation and the combat simulàtion. The events package is a common location for various information messages which are passed between entities in the simulation. The jlib package contains various utility or library classes. The maker package contains classes that are used set up and run the simulation based on an input data file. Finally, the supply package contains the classes that make up the heart of the logistics simulation. The event diagram for the supply package is shown in **Figure 2**.

Table 3 - FLEXLOGS packages

Package	Functional description
TM	The time master responsible for the scheduling and execution of simulation events.
Attribute	Tracks and logs the values of entity attributes.
Battle	A simple combat simulation.
Combat	The interface between the logistics simulation and the combat simulation.
Events	Classes that support information passing between objects in the simulation.
Jlib	Utility and library classes
Maker	Processes an input control file and runs the model.
Supply	The actual logistics simulation.

D. RUNNING THE MODEL

An input data file controls the operation of the model. The model can be run either on a single computer or in a client/server mode on multiple computers. Detailed instructions for running the model are contained in Appendix C. Two examples are given to illustrate the operation of the model. The output of the model is a file containing the values of any variables from the run along with the number of blue and orange wins. The output file is suitable for import into a statistics package such as Microsoft Excel or Math Soft's S-Plus.

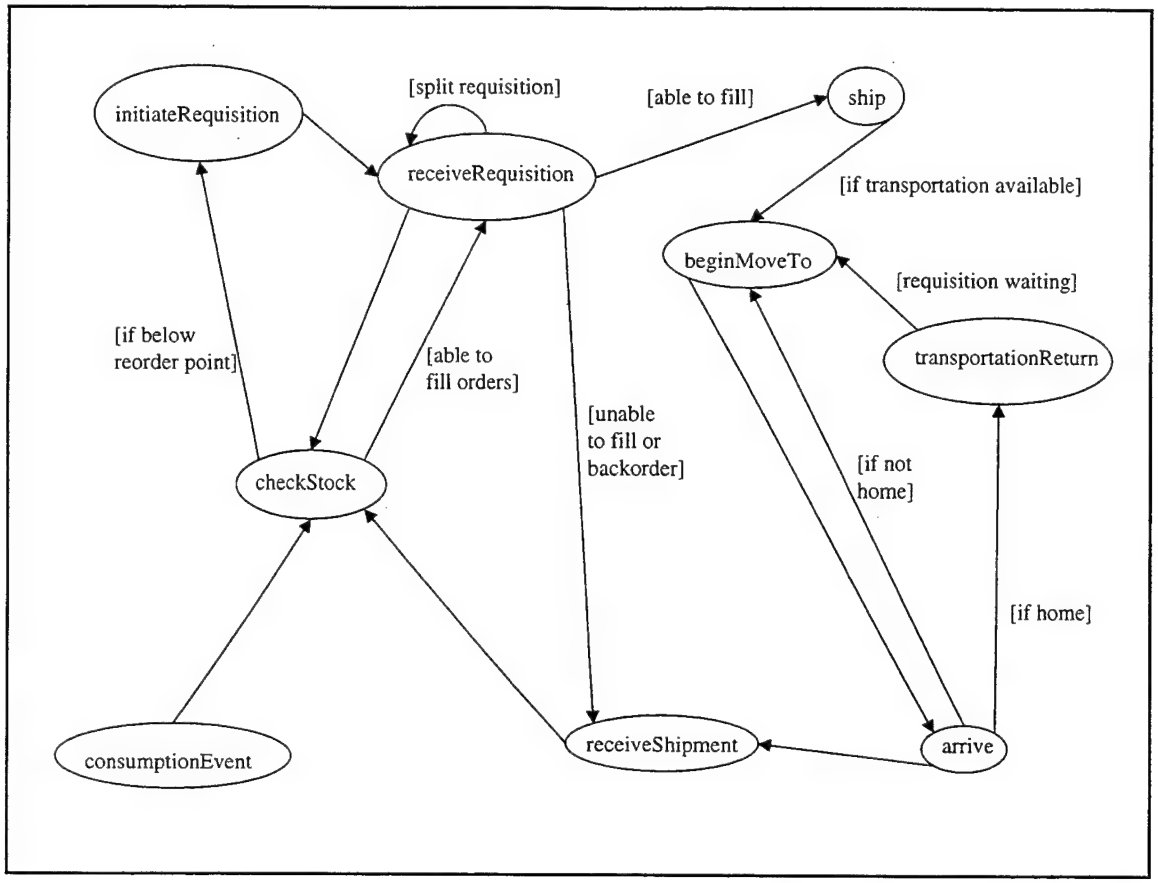


Figure 2 Supply Package Event Diagram – This figure shows the relationship among the simulation events for objects contained in the supply package. The event diagram documents the event scheduling logic contained in the simulation.

IV. AN ANALYSIS OF THE EFFECT OF LOGISTICS ON THE OUTCOME OF COMBAT

A. METHODOLOGY

1. Exploratory Analysis

The purpose of this research is not to predict the precise effect of exact logistics parameters on a particular battle, but rather to explore the general effects of logistics. Exploratory analysis is a term described in a number of publications by the RAND corporation [Banks, 1992] [Brooks, Banks, and Bennett, 1997]. These publications describe exploratory analysis as an alternative to the normal use of models in studies. Normally, models are used in a predictive manner. Furthermore, it is typically assumed that the model can be validated, so that for given input data, the output of the model is an accurate prediction of reality. These predictive studies usually focus in on a narrow range of optimal solutions.

What RAND proposes, and the approach followed in this research, is to use the model to explore a large solution space in order to get an idea of how the parameter under study behaves in a broad range of conditions. In this research it is important to explore this broad area since even the general nature of the relationships among logistics factors and combat effectiveness are not known. Another reason for the need for exploratory analysis is the inherent stochastic nature of systems such as combat or even simple

transportation networks. An optimization approach would typically emphasize a region around an "optimal" solution based on single expected values of the input variables. The method of exploratory analysis pursued in this thesis allows the model to directly take into account the stochastic nature of the problem and to identify regions in the solution space where changes in the outcome of a battle may more dependent on chance than on small changes to the input variables.

2. Measures of Effectiveness

Several measures of effectiveness (MOE) were considered for use in evaluating the effect of logistics on combat using the simulation developed by this research. One of the measures considered was the amount of time a combat unit in the simulation has one or more supply items below some designated critical level. While this MOE is easy to measure in the model and easy to interpret, it has a number of shortcomings. First, the critical level of supply is an arbitrary choice by the modeler (or in real life, by the commander.) An invalid choice of this parameter could cause the MOE to be meaningless. Second, the fact that a particular unit's supply falls below the critical level could have no impact on the outcome of the war.

Superior measures of the effectiveness of a combat logistics system involve how well the system supports combat rather than how well the supply system acts as a supply system. For these reasons, the MOE chosen for the analysis portion of this research was the number of times the Blue force wins the simulated war. This choice of MOE forced

the development of a simple combat model not only to generate the demands for the supply system, but also to determine the winner of the war.

3. Choice of Input Variables

One of the goals of Focused Logistics is to reduce the amount of material in theater. Thus, the amount of supplies stocked at combat units was deemed a good choice for an independent variable for the model. In order to reduce the logistics footprint under Joint Vision 2010, something must be substituted in place of the stockpiles of material. One of the substitutes mentioned in Focused Logistics is the improvement of information technology to aid the logistics system. However, the time available to develop the model did not allow for a measure of information technology to be included in the model.

Another available substitute for having material in place, is the ability to rapidly ship the material to where it is needed. Therefore, another substitute for material in place is premium transportation assets. The modeling of the number of certain types of transportation assets was well within the scope of the model developed in the allotted time and therefore is a good choice as the second independent variable.

The model analysis conducted as part of this research will explore the effect of varying stockpile levels and the availability of premium transportation, specifically airlift on the outcome of the simple combat simulation.

B. RESULTS

The first analysis conducted uses a combat model that simulates a simple battle. Units on each side advance in the direction of the enemy objective. If a unit detects an enemy unit, the units fight. When a unit reaches its enemy's objective, that unit's side is declared the winner. Demand for supplies is generated when units fight. While this combat model shows the relationship between stockpile size and combat effectiveness, the effect of premium transportation is not demonstrated. An investigation into the model reveals that the typical combat unit in the simulation is engaging the enemy less than once every four days. Therefore, the unit can easily be re-supplied by slow transportation prior to the next time the unit engages in combat. Since modifications to this combat model did not produce any increase in combat frequency, another combat simulation, described below, was developed to allow direct control over the intensity of combat.

The newly developed combat model is designed to allow direct control over combat frequency. Each unit has a probability of fighting once per period. Both the probability of fighting and the length of the period can be set when the units are constructed. If a unit fights, it consumes a randomly determined amount of supplies. A more detailed description of the combat model is contained in Appendix B.

The logistics network used to develop the relationship between combat effectiveness and stockpile size and amount of premium transportation is shown in Figure 3. In the simulation runs used, there were a total of 8 combat units. Each combat unit had a 50% probability of fighting about once every 24 hours. These times were normally

distributed with a mean of 24 hours and a standard deviation of 0.5 hours. If the unit fought, it consumed approximately 2000 cases of bullets. The amount consumed was normally distributed with a mean of 2000 and a standard deviation of 100. If the blue unit tried to fight while below the critical supply level, then blue was declared the loser. If the simulation ran for 30 days without blue losing, then blue was declared the winner. Sufficient sealift and land transportation was provided so these modes were not a limiting factor.

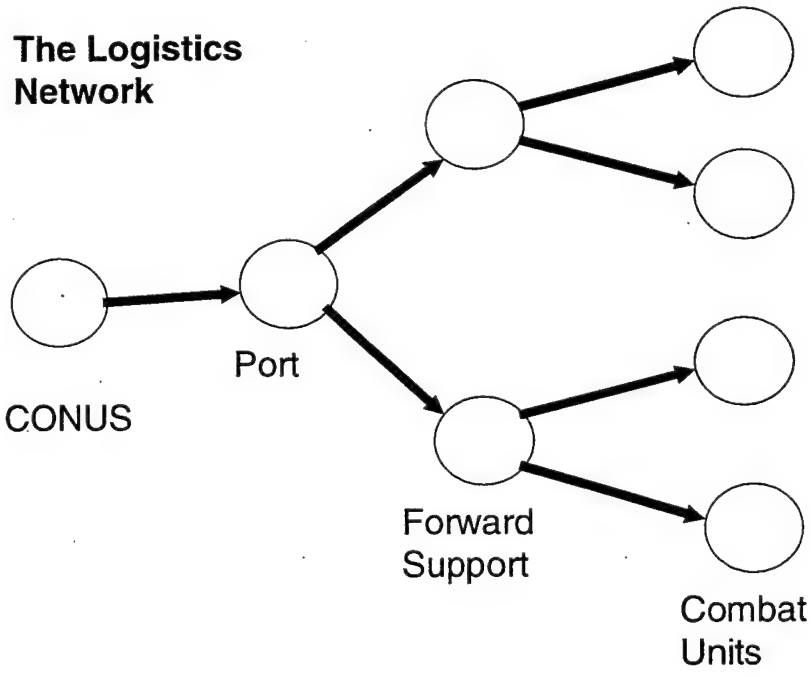


Figure 3 Simple Logistics Network - This network was used to generate the data necessary to study the relationship between combat effectiveness, stockpile size, and amount of premium transportation. The figure shows the supply ordering relationships between the units in the simulation. The demand for supply is generated from combat by the Combat Units.

Figure 4 shows a three dimensional perspective of the curve generated. The same curve is shown as a set of color-coded level curves in Figure 5. In both figures, blue represents the region where the blue side always wins. The red represents the region where the blue side always loses. Other colors represent the transition region where each side won some of the battles. On the level curves it is worth noting that at a high enough level of stockpiles, blue can win without use of premium transportation. As the amount of premium transportation available is increased, the size of the stockpiles required decreases. This appears to be a near linear relationship until airlift reaches the saturation point where additional airlift does not reduce the stockpile requirements.

Next, in order to determine the effects of a different logistics scheme, the simulations were re-run with a slightly modified logistics scheme. If a unit (a combat unit or forward support unit) is unable to get supplies from its normal supplier, it is allowed to get them from the closest unit that has supplies available. Since the relaxed rules would allow a unit an alternate source of supply, the author originally believed that this relaxation would allow the level curves to shift down and to the right so that the same level of combat effectiveness would be achievable with smaller stockpiles and less premium transportation. However, the curves generated with this modified system were not significantly different from the original curves.

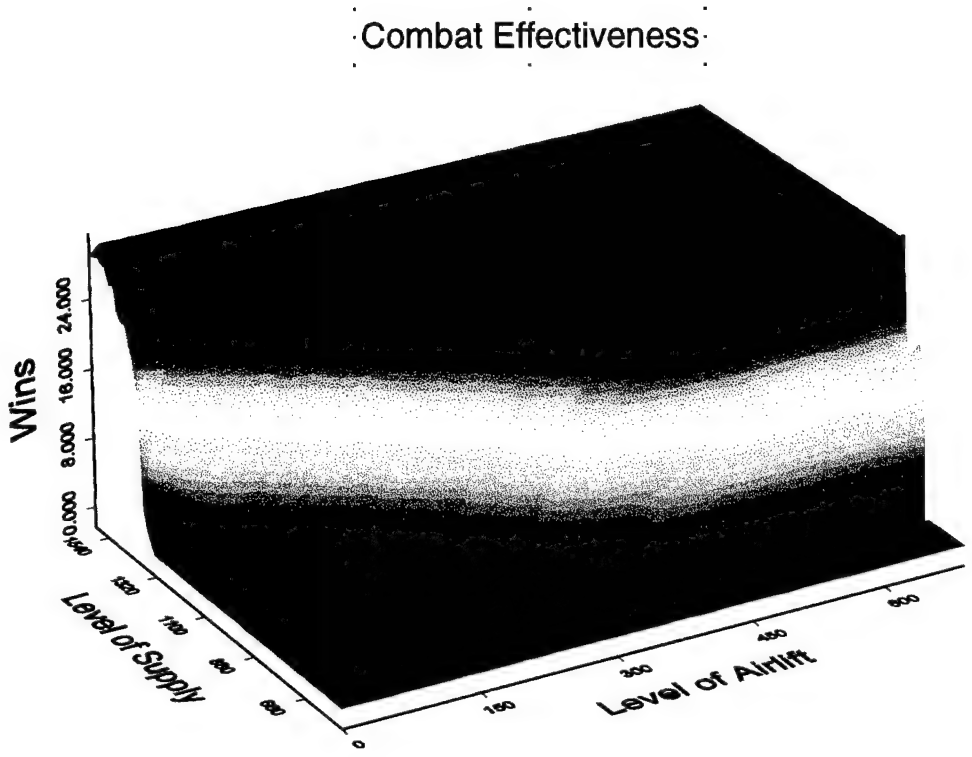


Figure 4 Combat Effectiveness Curve – This figure illustrates the effect of amount of airlift available and level of supplies in theater on the number of times the blue force wins. The blue region represents the region of the graph where blue always wins. The red region represents the region where red always wins. Other colors represent the transition region where each side wins some of the battles.

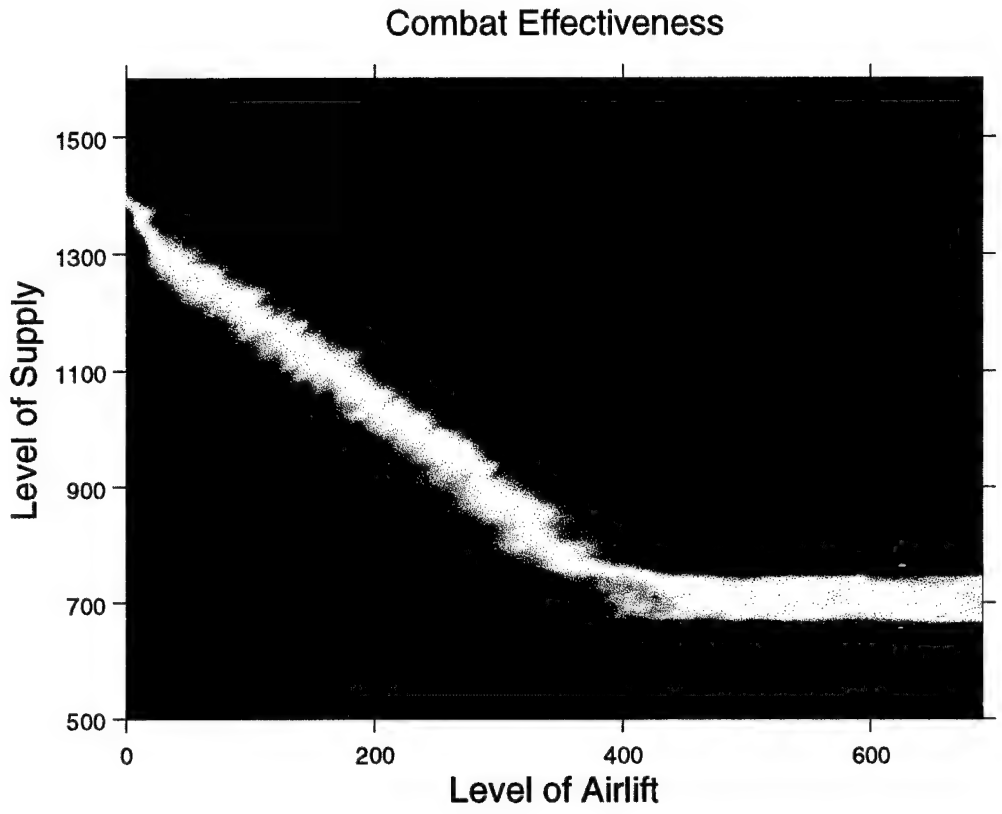


Figure 5 Combat Effectiveness Level Curves – This figure is a color-coded level curve representation of the same data as the previous figure. As in the three dimensional graph, the blue region is where blue always wins, the red region is where red always wins, and other colors are where both sides win some of the battles. The graph demonstrates that blue can always win even without premium transportation if enough supplies are present in theater. As the amount of premium transportation is increased, there is a near linear decrease in the amount of supplies needed to win until the airlift reaches the saturation point. Addition airlift beyond this saturation point causes no further reduction in the amount of supplies needed.

In order to determine the cause of these unanticipated results, the stock levels for each type of supply node over the length of a typical battle were examined. Figure 6 shows the stock level at a typical combat unit in the case where the blue side always wins. The horizontal line at 2000 represents the stock level that is considered critical. The supply level drops below the critical level whenever the unit fights, but is quickly restored above this level prior to the next combat.

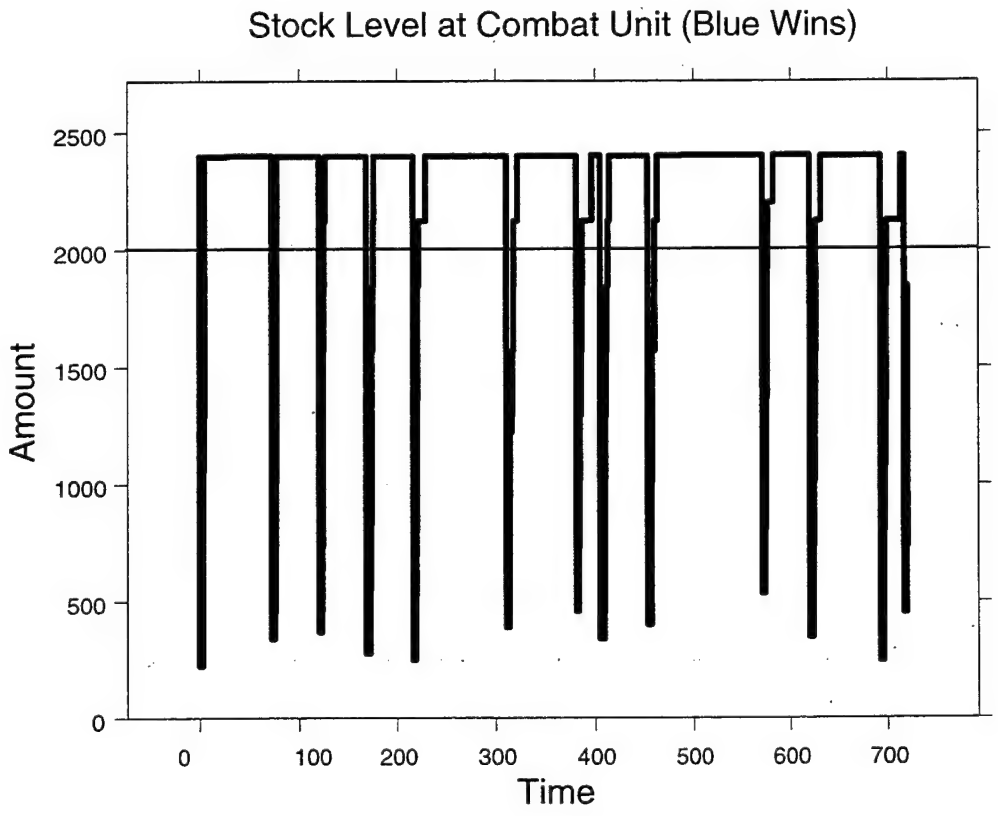


Figure 6 Combat Unit Stock Levels (Blue Wins) – This graph shows that in the case where the blue side wins, the stock levels drop below the critical level, but the unit is resupplied prior to the next combat.

Figure 7 shows the stock level at a typical combat unit in the case where blue always loses.[2] This graph is not greatly different from the case where blue won, except that the time that the stock level is below the critical level increases, slightly as the war goes on.

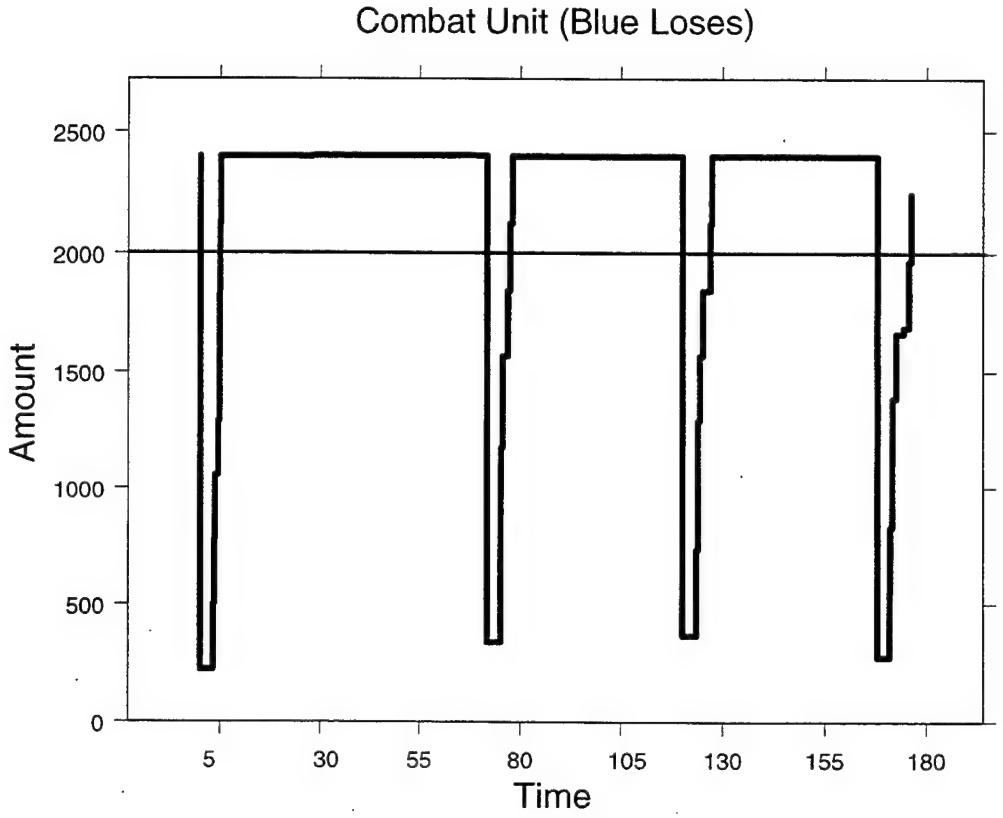


Figure 7 Stock Level at Combat Unit (Blue Loses) – The graph for the case where blue loses in not drastically different from the previous case, except that the time below the critical level is increasing as the war progresses.

---

[2] Note the change of time scale between the case where blue wins and where blue loses since the data logging stopped as soon as blue lost.

Since the problem is not obvious from examining the stock levels at the combat unit, one of the forward support bases will be examined. Figure 8 shows the stock levels at the forward support base for the case where blue wins. The graph shows the levels appear to stay safely above the zero level. Whenever levels dip close to zero, they are quickly replenished by airlift.

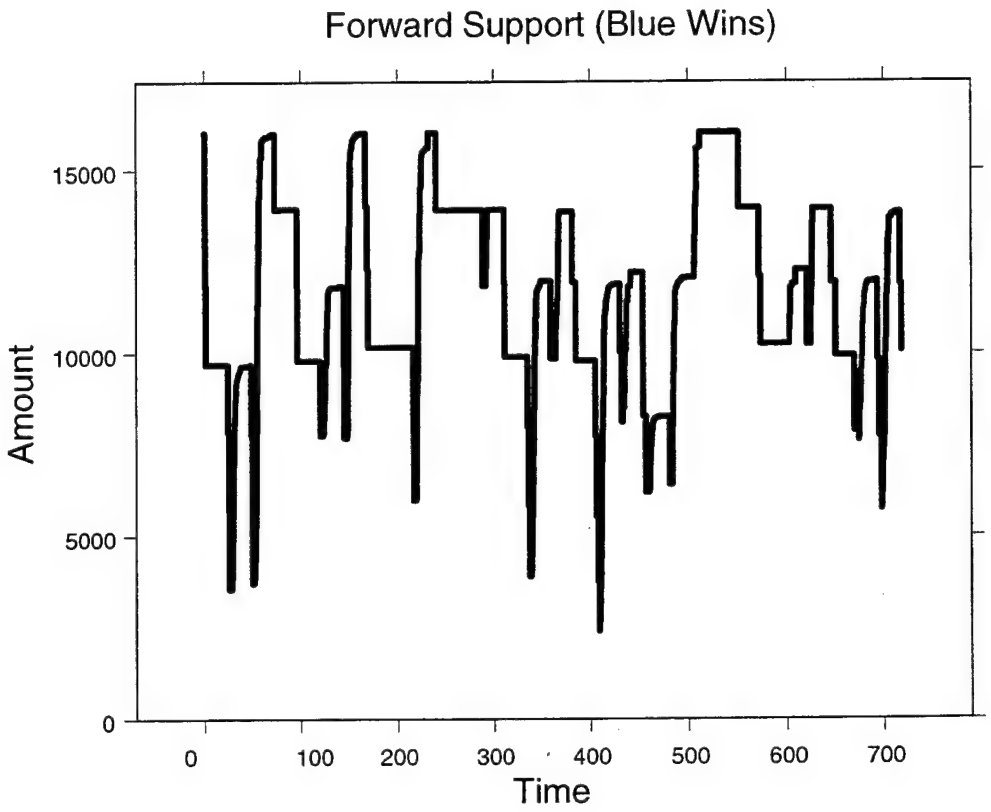


Figure 8 Stock Level at Forward Support Base (Blue Wins) – This graph shows that in the case where blue wins the stock levels at the forward support base stay safely above zero.

For the case where blue loses, Figure 9 shows the stock level at a forward support base. In this case, the stock levels do reach zero, but they recover quickly. There is not obvious cause of the logistics system failure at this level.

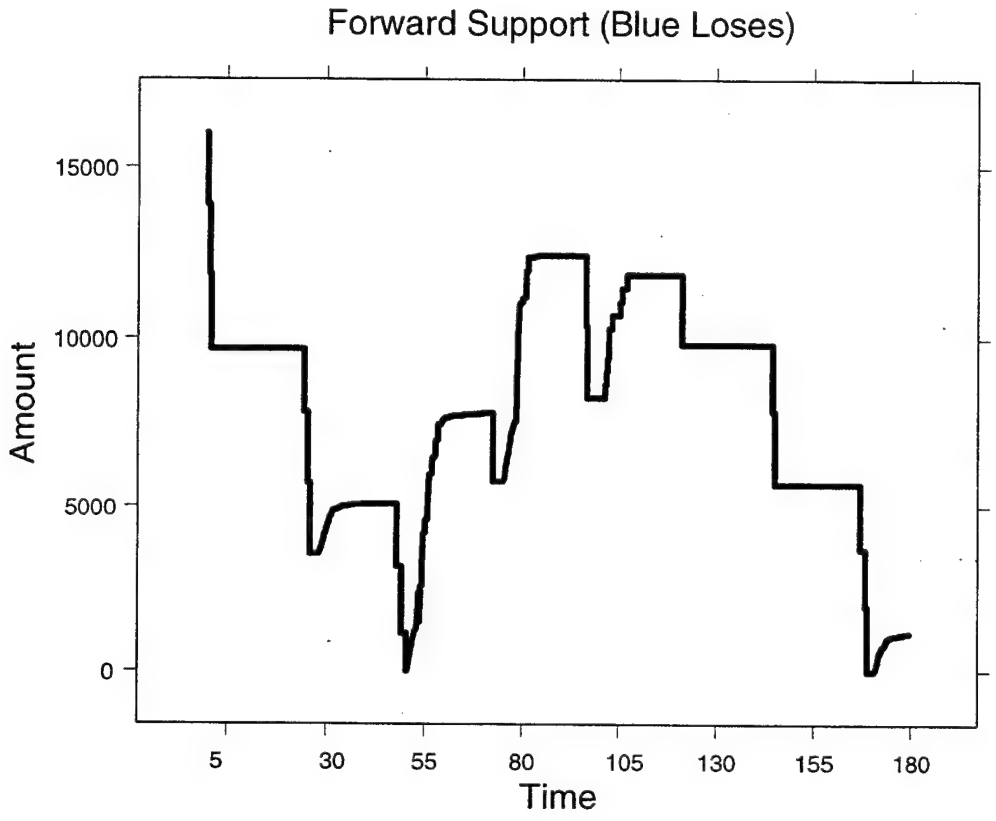


Figure 9 Stock Level at Forward Support Base (Blue Looses) – The stock levels at the forward support base for the case where blue looses show that while the levels do reach zero, they quickly recover.

Figure 10 shows the stock levels at the port of debarkation for the case in which blue wins. This graph shows that airlift is able to maintain stock at a usable level until the sealift arrives. Figure 11 shows that in the case where blue loses, the stock level at

the port is quickly used and there is insufficient airlift available to replenish the port while waiting for the sealift to arrive.

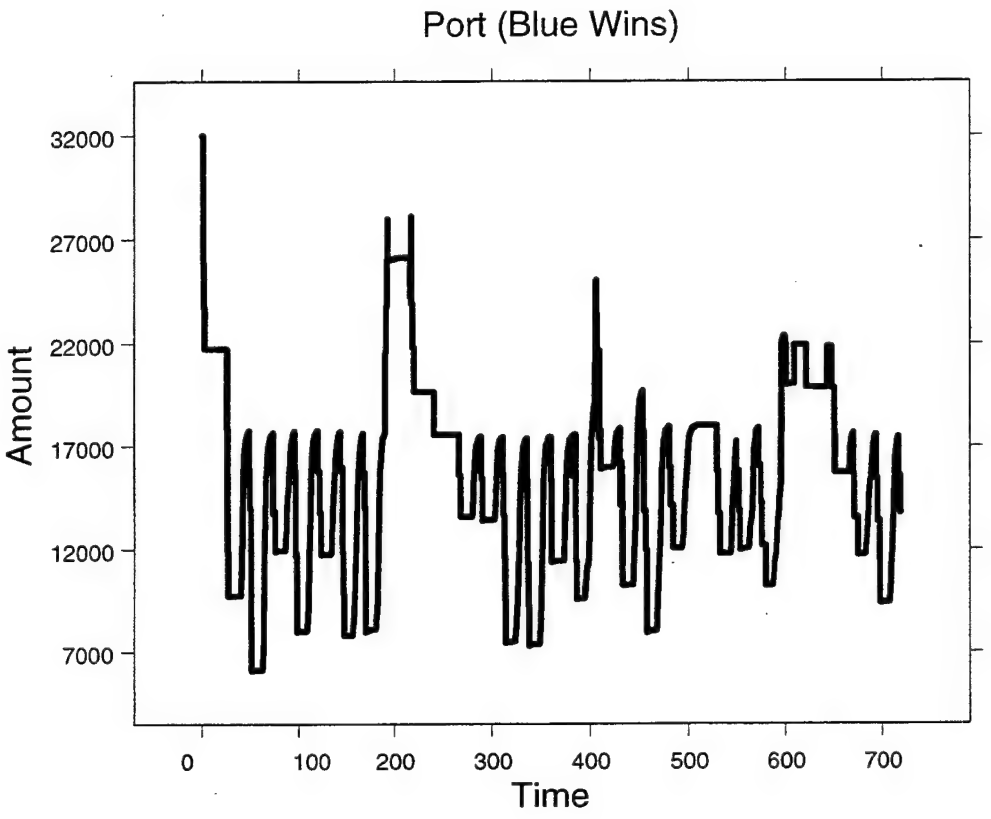


Figure 10 Stock Level at Port (Blue Wins) – This graph shows that airlift is able to maintain the stock levels until the arrival of the supplies via sealift.

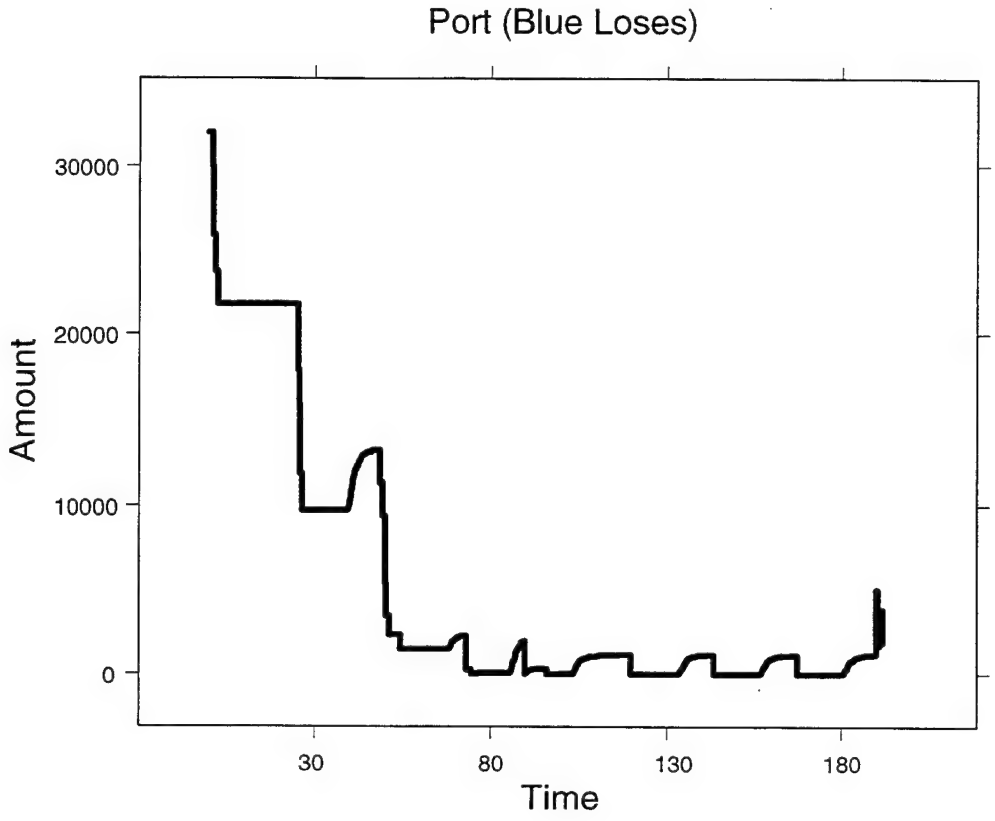


Figure 11 Stock Level at Port (Blue Loses) – This graph illustrates that without sufficient strategic airlift, the supply level at the port quickly reaches zero causing a stock shortage throughout the theater.

This series of graphs show that the cause of blue losing was not an intra-theater transportation problem, but rather a problem with lack of strategic airlift. This explains why the relaxation of the intra-theater supply rules had no effect on the amount of airlift and stock required for success.

Next, to further investigate the nature of the effect of a change in supply policy in the theater, the model was modified by adding sufficient strategic airlift and only varying

the amount of in theater airlift. Figure 12 is a contour plot of combat effectiveness for the case where there is unlimited strategic airlift and normal re-supply rules. Figure 13 shows the results when the re-supply rules are relaxed to allow units to order from nearby units when unable to get supplies from their normal supplier. These curves show that there is little change in the results by allowing the relaxed re-supply rules, however the relaxation did appear to increase the variance of the results. This indicates that in the scenario under study, the relaxation has no benefit and may cause the outcome of the battle to become less certain.

This chapter demonstrated how FLEXLOGS could be used to show the relationships between combat effectiveness, stockpile size, availability of premium lift, and supply system rules for a specific scenario. This section also demonstrated the ease with which the model could be modified and data gathered to allow not only the original question to be explored, but also analysis of issues raised when the results of the original study were counterintuitive.

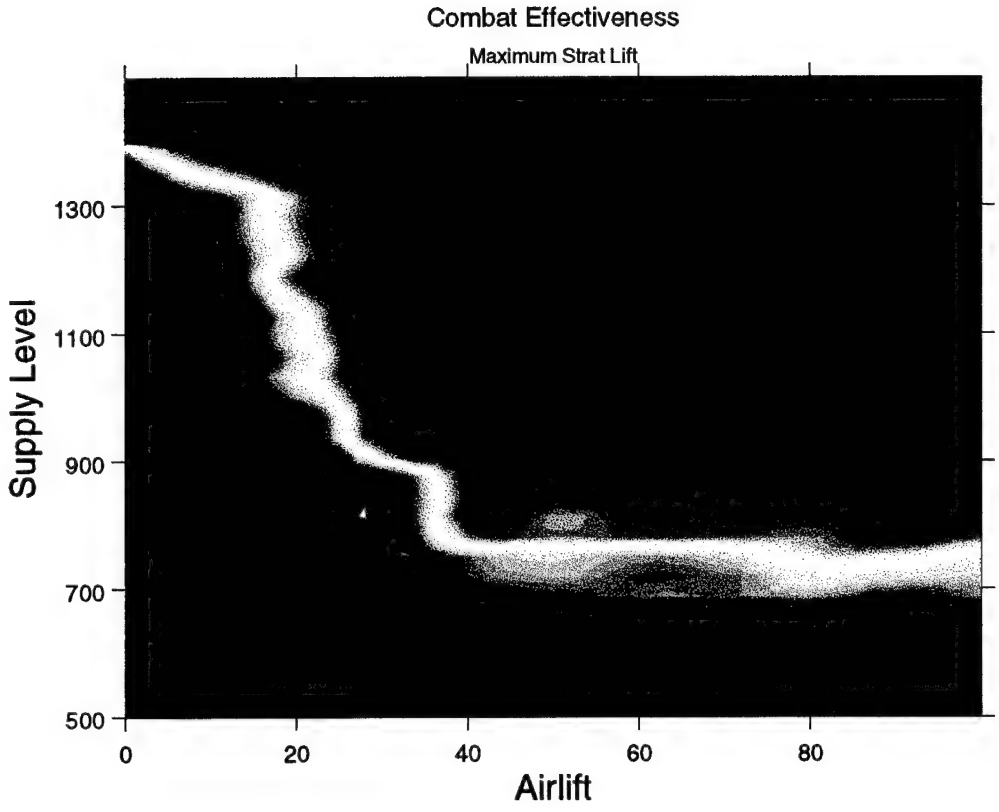


Figure 12 Combat Effectiveness with Unlimited Strategic Lift – This curve shows the combat effectiveness when unlimited strategic airlift is available. Blue indicates where blue always wins, red indicates where red always wins, and other colors indicate the region where both sides win some of the battles. The shape of this surface is very similar to the surface generated by varying the amount of all types of airlift.

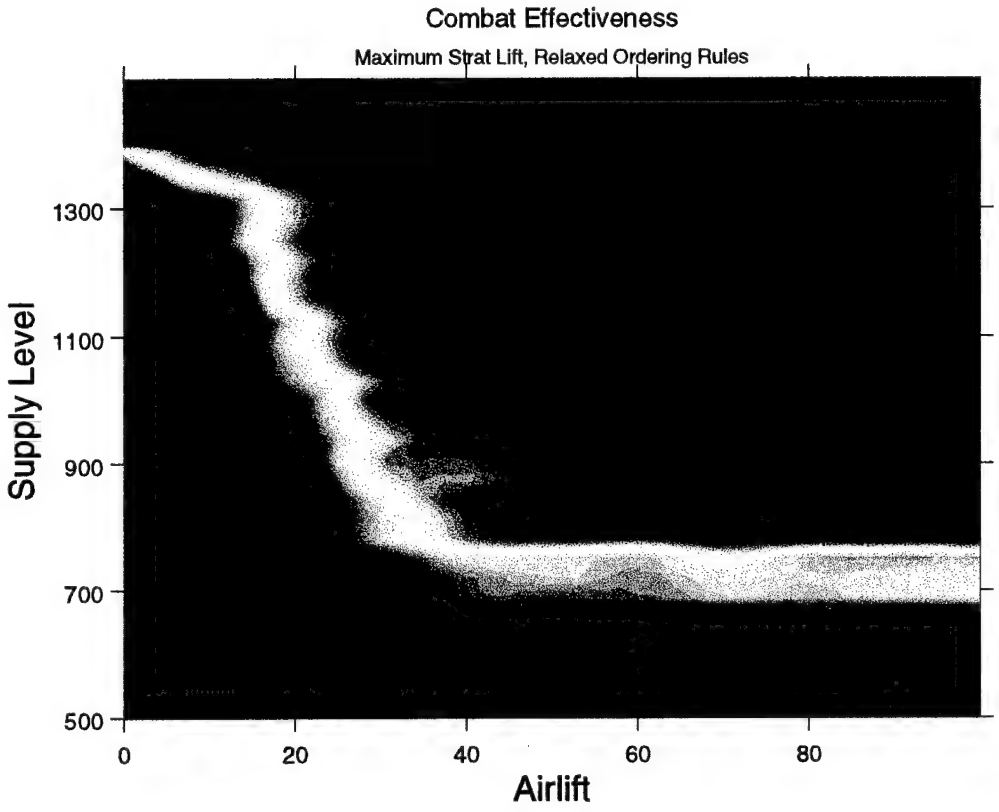


Figure 13 Combat Effectiveness with Unlimited Strategic Lift and Relaxed Supply Rules – This curve shows the combat effectiveness when strategic airlift is unlimited and units are allowed to order from nearby units if they are unable to be supplied by their normal supplier. The shape of the curve is basically the same as the previous case without relaxed supply rules, although the region where neither side always wins is larger. This comparison shows that, in the scenario studied, the relaxation of supply rules does not improve combat effectiveness and makes the outcome of the battle less certain.

V. CONCLUSIONS AND RECOMMENDATIONS

This research has developed a useful tool for exploring the future of military logistics. FLEXLOGS provides this tool through flexible design. By virtue of this flexibility, a wide range of potential future logistics systems can be studied to see their effect on combat effectiveness. One of the keys to this flexibility is the use of loose coupling to the combat model or other model used to generate the demands on the logistics system.

In this thesis, the nature of the relationship between stockpile levels, premium transportation, and combat effectiveness has been analyzed. In the course of analyzing the scenarios used for these demonstrations, the ability of the model to be rapidly modified to explore new questions was clearly shown. Data needed to explain counterintuitive results were easily obtained from the model.

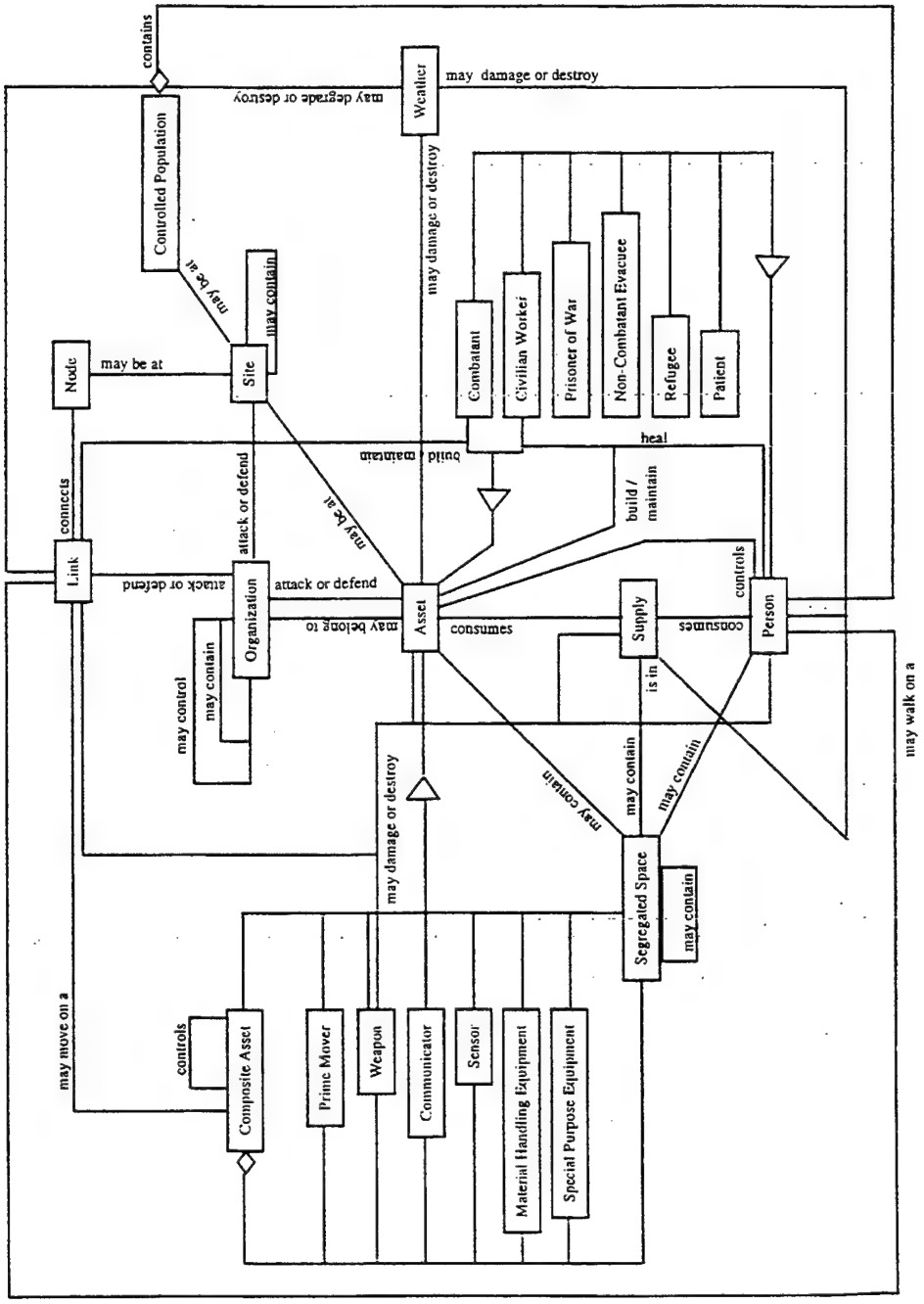
The model presented in this thesis is only a starting point. Appendix G contains a list of potential improvements that were identified during the development of FLEXLOGS for which there was insufficient time to include in the model. These improvements are of a general nature and should be prioritized and added to the model by future researchers. Additionally, the exact nature of the studies to be done using the model should be identified so that the model can be evaluated and, if necessary, modified.

This thesis demonstrates that a flexible simulation could be developed from a conceptual model with a relatively small level of effort. Also, by following sound object-oriented design techniques, the resultant tool can be made very flexible.

FLEXLOGS provides a flexible tool for the analyst to explore what Focused Logistics means to our forces. The model also demonstrates that a combat model and a logistics model can be integrated in a way that allows the combat model to be replaced with a minimum of effort.

APPENDIX A CONCEPTUAL OBJECT MODEL DRAFT ENTITY RELATIONSHIP DIAGRAM

High Level Object Model Of Military Logistics



[Draft COM, 1997]

APPENDIX B OBJECT DESCRIPTIONS

This appendix provides a detailed description of the objects in each package of the simulation. In the following paragraphs, the actual class names are indicated by bold type.

1. The TM package

The TM package contains the classes that are responsible for the scheduling and execution of the simulation. The classes in this package obviously have no counterparts in the conceptual model. Additionally, this package contains a generic priority heap class (**Heap**) which is used in the simulation both as the foundation of the time master and for holding lists that must be prioritized. (e.g. A heap for backorder processing).

The interface **Scheduler** must be implemented by any object that will have events scheduled with the time master. The time master is made up of the classes **TimeMaster**, **EventHeap**, and **JEvent**. Each object in the simulation that schedules events must have a reference to the instance of time master. The **TimeMaster** class contains an **EventHeap** that contains the events to be processed (**JEvents**). The **TimeMaster** class has methods that implement various ways to schedule new events and to cancel pending events. Once the initial events are scheduled, the simulation is started by calling the start() method of the **TimeMaster**. The **TimeMaster** exits, stopping the simulation, when either no more events are waiting on the heap, a specified number of events has been executed, or an object calls the flush() method emptying the event heap.

The classes **Heap** and **HeapEntry** are used as the basis for the **EventHeap** (which is why they were placed in this package and not elsewhere.) This generic priority heap is also used in other classes to store entities that require sorting by some priority, such as backordered requisitions.

2. The attribute package

The attribute package consists of classes that are used to track the values of attributes of the various objects in the simulation. Some examples are the number of personnel assigned to a unit and the amount of each type of supply held in stock by a unit, and other such information. The purpose of storing this information in the attribute classes is to allow the values to be tracked outside of the object that holds them. The package also contains classes that are used to write the value of these attributes to a log file for later analysis.

The values of the attributes themselves are stored by the classes **IntAttribute** and **DoubleAttribute**, derived from the abstract class **Attribute**. All **Attribute** objects are identified by a name, given when the object is constructed. These classes also contain methods to set and get the value of the attribute.

The value of an attribute is made visible to other objects in the simulation by implementing an event-listener model. The event-listener model is discussed in the events package. **Attribute**, and its subclasses, are the source of **AttributeChangeEvents.** Any object that requires notification of an attribute change, implements the **AttributeChangeListener** interface and then must register with the **Attribute** in order to

be notified of the change. The class **DataLogger** in the attributes package implements the listener interface and records the values of the attribute for which it is registered to a data file for later analysis. The **DataLogger** can also record the values of a **CoordinateAttribute**, which is derived from the **Coordinate** class, used to track the location of an object. Both the **Coordinate** and the **CoordinateAttribute** are contained in the jlib package.

In order to make management of a large number of attributes in a class easier, the **Attributes** interface was developed. Any class that has a number of attributes should implement this interface. The methods of the **Attributes** interface allow an object to gain a reference to an attribute by calling the getAttribute() method and referencing the attribute by its name. An implementation of the **Attributes** interface, **AttributeContainer**, is also provided in the attribute package. A class with many attributes can have an **AttributeContainer** in order to delegate attribute management. This class, in addition to the methods from the **Attributes** interface, has methods to add attributes to the container.

The final class in the attribute package is the **WinLogger.** This class is a **WinEventListener** (an interface in the events package) and logs the results of a war to a file, recording the winner and the length of the war.

3. The battle package

The battle package contains classes that provide the infrastructure for a combat model, although the actual implementation of the model is contained in the combat package.

Any class whose instances will participate in the war simulated by the battle package must implement the **Combatant** interface. This interface defines the minimum combat related events that a combatant must recognize. It also contains methods that provide the information necessary for interaction with the **Referee** class. In addition, any class implementing the **Combatant** interface must have a method that provides a homogenous Lanchester aimed fire coefficient used to resolve the outcome of battles.

The **Referee** class contains the logic necessary to adjudicate the detection of objects by each other on the battlefield. It is responsible for determining the detection times for all combatants in the simulation. The **Lanchester** class contains methods that are used to calculate the length of a battle along with remaining force strengths. This class uses simple homogenous force, aimed fire Lanchester equations for calculations. [Taylor, 1983] A more complex method of calculation was not chosen since the combat portion of this simulation was added simply as a way to generate logistics demands and to provide a measure of effectiveness for logistics. Based on these calculations, the results of a battle are stored in a class called **BattleResults** for later use. This class was necessary since the results of the battle are calculated at the beginning of the battle, but are not needed until the conclusion of the battle to update the state of the entities

involved. In order to provide an ending point for a war, the **Objective** class was developed. **Objective** implements the **Combatant** interface so that other combatants can detect it. When a combatant detects the **Objective**, that combatant is declared the winner of the war.

The final class in the battle package is the **OrangeLogistics** class. This class is used to take the place of a complex logistics model for any combatant that is considered unconstrained by logistics in the simulation. It implements the **Logistics** interface in the supply package, however, any query of stock levels simply returns a number indicating an unlimited quantity.

4. The combat package

The combat package is the interface between the logistics and combat simulations. It contains interfaces that provide the logistics simulation with its view of the combat simulation. The class that implements the combat simulation is also contained within this package.

Any class that will be used to generate the demands seen by the logistics simulation will need to implement the **CombatUnit** interface. This interface is how the logistics simulation communicates with the combat simulation. A **CombatUnit** is the source of **ConsumptionEvents**, to be discussed later in this section, and therefore extends the **ConsumptionEventSource** interface. A **CombatUnit** also processes **CriticalSupplyEvents**, which is discussed in the events package. Finally a **CombatUnit** is a **Scheduler**, part of the TM package, since it can have simulation events scheduled.

The basic coupling between the logistics and combat simulation is through the passing of **ConsumptionEvents** by the **CombatUnit**, indicating that an amount of a certain supply has been consumed. The **CombatUnit** is informed when a supply has reached a critical level by receiving a **CriticalSupplyEvent.** The details of the different types of critical supply will be discussed in the events package.

The **CombatUnitImpl** class provides a basic implementation of the **CombatUnit** interface, including the methods needed for sending **ConsumptionEvents** and listening to **CriticalSupplyEvents**. Additionally, the **CombatUnitImpl** provides an event called consume() that once scheduled will cause a fixed amount of a single type of supply to be consumed every 8 hours. This feature was included for testing of the logistics model. The class **TestCombatUnit** extends this functionality by allowing the period to be changed and was also used during model development for testing.

Also included in this package is an implementation of a simple combat model. The design of this model is only briefly discussed since it is not the major thrust of this research and was only developed to produce logistic demands and to provide a win/loss determination for use as a measure of effectiveness. The **FlagCombatUnit** extends the **CombatUnitImpl** (and therefore implements the **CombatUnit** interface), implements the **Combatant** interface from the combat class, and the **WinEventSource** interface which will be discussed in the events package. The **FlagCombatUnit** will proceed in the general direction of its **Objective**. If it detects an enemy **FlagCombatUnit** (as determined by the **Referee**) it will stop and fight. The **Lanchester** class determines the

outcome of the battle. The winner continues moving to the **Objective** after a delay, the loser retreats away from the **Objective**. When a **FlagCombatUnit** detects the enemy **Objective** it stops the simulation and fires a **WinEvent** that indicates which side of the conflict won. Methods to generate any random variables needed by the combat simulation are contained in the **Dist** class. These distribution methods are contained in one class to allow easier modification of the distribution types.

StressCombatUnit is another implementation of the **CombatUnit** interface. This class was designed to give a higher combat tempo, and therefore higher consumption rates, than the other implementations provide. In the **StressCombatUnit**, each unit has a chance to fight once every time period. The probability of fighting and the mean period time can be set when an instance is constructed. If the unit fights, it consumes an amount of supplies chosen from a distribution that is normally distributed with a mean of 2000 and a standard deviation of 100. If a unit fights while a **CriticalCombatEvent** is active, then blue is declared the loser. If blue has not lost by 720 hours (which is currently hard coded in the constructor), then blue is declared the winner. The **StressCombatUnit** was developed to give a more complex demand pattern than the **CombatUnitImpl**, and to allow more control over the demand pattern than the **FlagCombatUnit**.

5. The events package

This package contains the various classes that are used to pass information between classes in a loosely coupled way. This method of information passing is called the event-listener model. In the event-listener model, there are two interfaces present.

The first interface is implemented by the source of the information and is the <*event name*>**EventSource** (where <*event name*> is replaced by the name of the event). The **EventSource** interface requires a method to add listeners to the list of objects to be notified of the event and a method to remove listeners. Also, the class implementing this interface must have the methods to actually create the events and notify the objects on the listener list. The <*event name*>**EventListener** interface requires a method that will be called by the **EventSource** when the event occurs. This method in the implementing class then takes whatever action is appropriate for the event. The final part is the <*event name*>**Event** class which is used to contain the information to be passed to the listener. The goal of the event-listener model is to allow the source classes to make information available to other classes without having knowledge of which classes may be listening until the method to add listeners is called at runtime [Buss, 1998 OA4333 class notes].

The first type of event defined in this package is the **CriticalSupplyEvent.** This event is used to notify a listener that a type of supply has reached a critical level. Two types of **CriticalSupplyEvents** are defined. The **CriticalMobilityEvent** indicates that the effected unit cannot move. The **CriticalCombatEvent** indicates that the level of a type of supply is a level that the unit should not initiate combat. What types of supply are connected to what type of **CriticalSupplyEvent** is determined when the supply types are created using the **SupplyType** class, which is part of the supply package.

The second type of event is the **WinEvent**. A **WinEvent** is generated when one side of a war is declared the winner. The event contains the winning side, the length of the war, and a remark. This event is used by a simulation to tally a win-lose scoreboard.

The final type of event, while modeled after the event-listener model, has a slightly different purpose. A means was needed to pass data between the combat and logistics models in a loosely coupled manner. The **InformationMessage** class and the two classes derived from it, the **DataMessage** and **RequestMessage** accomplish this task. One way these messages are used is for any class to call the getData() method of an **InformationSource** to request the current value of some attribute. The requestor passes the **InformationSource** a **RequestMessage**. The getData() method returns a **DataMessage** with the requested information. This method of getting information has the advantage of only requiring the requesting class to know that the class it requires information about is an **InformationSource**. The disadvantage is that the availability of the data cannot be checked until runtime. In this simulation the **InformationSource** is used by the **CombatUnit** to get information about the location and force strength from the **Logistics**.

Any class willing to accept unsolicited **DataMessages** implements the **InformationAcceptor** interface. This allows one object to update information in another object in a loosely coupled way. This is used in the simulation to allow the **CombatUnit** to update the position and force strength of the **Logistics**. The **InformationSource** and

InformationAccepter interfaces were used in this simulation to make a future conversion of the interface to the High Level Architecture (HLA) easier.

6. The jlib package

This package contains miscellaneous utility classes that are used in the simulation, but did not seem to fit cleanly in any other package.

The **Coordinate** class is used to store any two-dimensional position information used by entities in the simulation. It also contains methods for performing calculations on position data (e.g. the distance between two points) [Buss, 1997 – OA3302 class notes]. The **CoordinateAttribute** extends the **Coordinate** to allow **AttributeChangeEvents** to be generated whenever the position is changed. This is used for tracking the positions of the units in the simulation.

The **Quadratic** class contains a method to solve a quadratic equation and is used by the **Referee** in detection calculations [Buss, 1997 – OA3200 class notes]. The **Utils** class contains stand-alone methods used in various parts of the simulation.

The **RuckRandom** class extends Java's built in **Random** class to add more distributions.

7. The maker package

This package contains classes that are used to make running the model simpler. None of these classes are required to run the model, all of the objects needed could be instantiated in the main method of a Java class. The classes of the maker package allow the model to be controlled via an input control file. This section provides a brief

description of these classes; for information on how to set up a run the model using these classes see paragraph III.D above.

The **ModelMaker** class takes an input control file and in its constructor parses the file, creats the objects as directed by the input data, then runs the model. After the model run is complete, the winner of the simulation can be determined by testing the blueWin() and orangeWin() methods. The input lines are parsed using the **TokenReader** class. The **ModelDriver** class is used to conduct multiple runs of the model. The **ModelDriver** understands addition key words that allow the user to set up the number of replications, as well as vary any number of object parameters.

In order to be able to distribute the computations of the model over a number of networked computers, classes were developed to control execution of multiple runs over a network. The **ModelServer** class takes the same input control file as **ModelDriver**, but instead of running the **ModelMaker** sequentially on the local machine, it directs the execution on multiple machines by any number of **ModelClients** across a network.

8. The supply package

This package contains the classes that make up the main body of the logistics simulation. Table 4 is a summary of the interfaces in the supply package. The major classes of the supply package are shown in Table 5. The classes in this package represent a logistics system. Some of the classes represent a transportation network; others represent the organizations responsible for the various logistics functions. There are also classes that represent the information and material in the system.

Table 4 - Supply package interfaces

Interface	Description
Location	A place where an Asset can be located.
Logistics	An object that provides logistics functions to a **CombatUnit**.
MotorPool	Manages the transportation assets of a **SubOrganization**.

Table 5 - Supply package classes

Class	implements	extends	Description
Asset	none	none	Something that is used to accomplish a task.
Dist	none	none	Contains the random variants for objects in the supply package
Link	Location	none	A route between two Nodes.
LogisticsKnowledgeBase	none	none	A database that contains rules to relate suppliers to customers.
MotorPoolImpl	none	MotorPool	Manages **Transportation** for a **SubOrganization.**
Node	Location	none	A node in the transportation network.
Requisition	none	none	Represents both the request for supplies and the supplies themselves when in transit.
Site	None	none	A container for the network defined by the **Nodes** and **Links**.
SubOrganization	None	Asset	The logistics functions of a unit.
SupplyType	None	none	Defines the various commodities in the supply system.
TransType	None	none	Defines a unique type of transportation asset.
Transportation	None	Asset	A transportation asset.
VariableLink	None	Link	A **Link** whose distance depends on the location of the end **Nodes**.

The discussion of the supply class will start with a description of the classes that represent the logistics network. The **Site** class is used as a container for the logistics network. In the conceptual model, the **Site** could contain other **Sites** and **Assets** could exist directly in a **Site**. However, in the simulation implementation, the **Site** only contains the **Nodes**. The **Node** represents a point location in the logistics network. The location of the **Node** is determined by an instance of the **Coordinate** class. A **Node** may be moved during the simulation by changing the position of its **Coordinate**. The class

Link represents a path between two **Nodes** over which **Transportation** can move. The length of the **Link** is set when the **Link** is instantiated. **Links** can represent air routes, roads, sea routes, rail lines, and other types of routes. In order for a **Transportation** object to move on a **Link**, the transportation modes must be the same. A **VariableLink**, which is a subclass of **Link**, is a **Link** whose length is determined by the straight-line distance between its end **Nodes**. The **VariableLink** is used for routes between **Nodes** that will be moved during the course of combat. The **Node** class and the **Link** class both implement the interface **Location**. A **Location** is any object on which an **Asset**, to be discussed below, can exist.

The next group of classes in the supply package represents objects that are modeled as assets in the conceptual model. The parent object is called an **Asset**. The **Asset** represents things that are used to accomplish a task, but that are not consumed. An **Asset** exists at a **Location** and is on one side of the conflict (i.e. Orange or Blue). The first class derived from **Asset** is the **SubOrganization**. A **SubOrganization** is a unit that performs the logistics functions in the simulation.[3] In the simulation, the **SubOrganization** is the class that contains most of the events and logic for logistic actions. An event diagram for the supply package is shown in **Figure 2**. **SubOrganization** acts as direct logistics support to a **CombatUnit**, and can also represent supply sources not directly associated with a specific **CombatUnit**.

---

[3] The term SubOrganization is an artifact of the conceptual model, which contained a much more complex representation of military units.

SubOrganization implements the **Logistics** interface. The **Logistics** interface is implemented by any class that can provide logistics support to a **CombatUnit**. The **SubOrganization** stores information about the stock status of the unit, the number of personnel assigned, and the location of the unit.

The other class that is derived from **Asset** is the **Transportation** class. A **Transportation** object can move on a **Link** and carry filled **Requisitions.** Attributes of the **Transportation** asset, such as cargo capacity, speed, and transportation mode, can be fully specified when the object is instantiated so that the **Transportation** class can represent any type of transportation. In the current version (3.x) of FLEXLOGS an instance of **Transportation** can only carry a single **Requisition**.

The interface **MotorPool** is used to manage the transportation assets for a **SubOrganization**. The **MotorPool** is required to be able to store and construct **Transportation** objects, as well as to retrieve a **Transportation** entity suitable for shipping a given **Requisition** for the **SubOrganization**. These functions were implemented as an interface to allow the method of managing transportation in the simulation to be easily modified. The **MotorPoolImpl** class provides an implementation of the **MotorPool**. In this implementation, each **SubOrganization** has a number of **Transportation** assets assigned to it. When a **Requisition** is shipped, the **MotorPool** first selects the mode of transportation from a default list based on the priority of the requisition. The modes are checked in an order defined in the **SupplyType** class discussed below. The **MotorPoolImpl** selects a **Transportation** asset that is a

compatible mode for the **Requisition** starting with the largest available and working down until a minimum percentage of the capacity is used (currently 50%). If a single asset cannot carry the whole **Requisition**, then the **Requisition** is split and shipped in multiple **Transportation** objects.

A number of classes are necessary to implement the **MotorPool**. The **MotorPoolImpl** class stores the **Transportation** objects in an array of **ModePools**. A **ModePool** is all of the **Transportation** assets that are of the same mode (e.g. Air or Rail). The **ModePool** stores the assets in an array, sorted by size, of **SubMotorPools**. A **SubMotorPool** is all of the **Transportation** assets that are of the same **TransType**, in other words have the identical attributes. The **TransType** class holds a description of the identical assets. This number of classes to implement the **MotorPool** may seem large, however, the method of hierarchical storage was necessary to implement the logic to choose the desired size and mode of **Transportation**.

The **Requisition** class is used to represent both the data when a request for supplies is initiated and the material itself while in transit. The **Requisition** contains information such as the amount ordered, the minimum acceptable fill quantity, to whom the material should be shipped, the order time and priority, whether backordering is allowed, and the amount of material attached to the **Requisition**. Instead of limiting the simulation to an arbitrary list of commodity types, such as the ten supply categories, a **SupplyType** class is used to represent the different items in the simulation. This allows the level of detail of supply items to be controlled at run time. The **SupplyType** class

contains information about the item it is representing including, the volume per unit of issue, the density, a list of preferred transportation modes[4], whether the type is critical to combat or mobility, and the name of the item.

The final two classes in the supply package are part of the strategy for flexibility built into the model. The **Dist** class, like the **Dist** class in the combat package, holds all the methods for generating random variants for the events in the supply package. The **LogisticsKnowledgeBase** is a database that is used to determine from where a **SubOrganization** should order. The supply source is determined based on who is ordering, the **SupplyType** ordered, whether the order is high priority, and the amount ordered. Methods in **LogisticsKnowledgeBase** allow the supplier-customer relationships to be established at runtime so that different schemes can be tested easily.

---

[4] The current implementation of **SupplyType** contains a default mode priority list, which cannot be changed at runtime.

APPENDIX C INSTRUCTIONS FOR RUNNING THE MODEL

An input data file controls the operation of the model. This appendix discusses how a typical run should be set up and executed. The use of the model on a single computer will be covered in detail, followed by a section that explains the differences for use of the model in a client/server environment. The simulation runs used to gather the data for this thesis are used as examples.

The basic steps in running the model consist of setting up the input control file, executing the runs, then importing the results into a software package for analysis. The input control file consists of lines that begin with a "key word" followed by data needed to process that key word. Appendix C contains definitions and explanations of the parameters of all of the key words.

The first example is for a model that uses a **StressCombatUnit** as the source of demands. The input file discussed is show in Appendix E. In order for the construction of a model using the control files to be successful, the file must follow a logical ordering. Some of the lines in the file must be processed in a specific order since there is a dependency among some keywords for data. Some keywords may appear anywhere in the file. The paragraphs that follow will go through the sample control file line by line noting which lines must be in a specific order. The current version (3.x) of FLEXLOGS provides only limited error checking of the control file. The reader should reference Appendix E throughout this discussion.

The first line in the file causes the software to create the **Site.** This line must appear prior to any lines creating **Nodes** or **Links.** The description is required, but is only used as a label and can be any series of alphanumeric characters.

The "RUNS" line is typically next, but can appear anywhere in the control file. This line determines the number of replications for each data point. The next lines (VAR) also can be placed anywhere in the file. The "VAR" keyword is used to define the range and step size of variables in the model. Any data element that would normally require a number can be replaced by "*#varName#*" where *varName* has been defined by a VAR line. The rest of the data in the VAR line determines the starting value, the stopping value, and the step size for the variable defined. The starting value must be less than the stopping value and step size must be positive. Any number of VAR lines may appear in the control file.

The next line (NODES) constructs the given number of **Nodes** all at the origin. This command is useful when constructing a number of **Nodes** that do not move and that will be connected by **Links** whose lengths are fixed and determined ahead of time. The NODE line, to be discussed in the second example, is used to create **Nodes** at a specific location. All of the **Nodes** in the simulation should be constructed prior to any **Links**, **SubOrganizations**, or other entities located at a **Node**.

After all of the **Nodes** are constructed, the **Links** can be added between them. The keyword LINK is used for this purpose. The LINK line contains data for the two end **Nodes** of the **Link**, the transportation mode, the initial **Link** condition, and the length of

the **Link**. Some of the example LINK lines demonstrate that a comment can be added to the end of any line after all of the required data.

The next line in this example is used to define a **SupplyType**. The key word "SUPPLY" is followed by the name used to identify the **SupplyType,** the unit of issue, the weight per unit of issue, the volume per unit of issue, and finally the type of criticality for the item. In this example, we have defined a **SupplyType** "bullets" that have a unit of issue of "box." The bullets weigh 180 pounds per box. Each box occupies 1 cubic foot and the bullets are critical for combat. Note that the units of weight, distance, and volume are arbitrary but must be consistent. For example, if the volume of the supplies is given in cubic feet, then the volume capacity of transportation must also be in cubic feet.

A SUPPLYSET line is used to define a quantity of supplies that will be used when constructing **SubOrganizations**. All of the SUPPLY lines must appear prior to defining any SUPPLYSETs. A SUPPLYSET line creates an array of quantities whose length is equal to the number of **SupplyTypes** defined. Any SUPPLY lines that appear after the SUPPLYSETs will cause errors later in the control file processing. The SUPPLYSET line provides a name of the set, followed by a default quantity, followed by pairs of the names of **SupplyTypes** and quantities for that type. The line ends with the word "ENDSET" to indicate no more **SupplyTypes** follow. Any **SupplyTypes** not specified in the line are assigned the default value. In this example, since only one **SupplyType** exists, only the default value is used. Supply sets can also be derived from other supply sets. This is accomplished using the key word "SUPPLYSETMULT." The SUPPLYSETMULT creates a new supply set by multiplying the values of a previously

defined set by a scalar factor. For example the line "SUPPLYSETMULT/1sup/1dos/2.5//" creates the set "1sup" by multiplying the values in "1dos" by 2.5. The line that begins with "REM" is a remark and is ignored by the program. Note that since the variable substitutions are made prior to the file being processed, the software will attempt to substitute for any variables of the form "*#variable#*." To prevent an error caused by an undefined variable, the '#' was changed to "*" so that the whole line would be ignored.

The next group of lines creates the **SubOrganizations** for the simulation. The "SUBORG" or "SUBORGANIZATION" line provides the name of the **SubOrganization**, the number of the **Node** on which it is located, the supply sets that define the various supply levels, and the personnel strength of the unit. The meaning of the six supply sets is contained in the full description of the SUBORG line in Appendix C. The **SubOrganizations** must be defined prior to any **CombatUnits** that depend on them for logistics support.

The next entities defined in the input control file are the transportation assets. The TRANS or TRANSPORTATION line creates the given number of assets at the specified **SubOrganization**. This control line contains data that determines the transportation mode, the capacities, and the speed of the **Transportation**.

The DEFAULTSUPPLIER lines determine the source of supplies for each **SubOrganization**. These lines must appear after the SUBORG lines. The line simply specifies whom a **SubOrganization** should order from if there is no other ordering

information in the knowledge base. In this example no other ordering information is specified.

This example uses a **CombatUnit** as the source of demands for the simulation. The "SETDEFAULTCOMBATUNIT" line specifies the name of the class that will be the **CombatUnit**. The name of the class must be given in its fully qualified form as shown in the example. The class must implement the **CombatUnit** interface and have the correct constructor. Once the **CombatUnit** is set, the next lines create **CombatUnits** and schedule their initial events. The COMBATUNIT line creates a **CombatUnit** with the given **SubOrganization** or **OrangeLogistics** as its source of logistic support. The COMSUME lines schedule the consume event of the **CombatUnit** just constructed at the indicated time.

The next two lines are used for debugging and should be commented out for multiple runs. The DUMP line causes the **ModelMaker** to produce a dump file containing the state of the model prior to starting the **TimeMaster**. The SETDEBUG line causes the debug flag to be set for all classes. This will produce a very large amount of output to the console for this model.

Finally, the RUN line starts the model. The number specified limits the length of the run. Any lines after the RUN line will not be processed prior to running the model and may cause errors.

This example showed the basic steps needed to run a model with a simple **CombatUnit**. The example that follows uses the **FlagCombatUnit** and is a more complex model to run. In the paragraphs that follow only the differences from the first

example will be discussed in detail. The control file for this example is also in Appendix E.

The first difference in this example is the creation of some of the **Nodes** at specific locations using the NODE line instead of the NODES line. The NODE lines specifies the two dimensional location of the **Node**. The **Nodes** are still numbered in the order they are created.

The next difference is the creation of **Links** using the VARIABLELINK key word. This key word accomplishes that same function as the LINK key word, however the length of the **Link** is not specified in the control file. The length of a **VariableLink** is calculated during model execution as the straight-line distance between the two end **Nodes**. This line is useful for routes between **Nodes** that move during the simulation.

Since in this example, the **CombatUnit** is a **FlagCombatUnit**, some specialized key words are used. The OBJECTIVE lines set the location for the combat objective for both the orange and the blue side. These lines must appear before any of the **FlagCombatUnits** are created.

The ORANGELOGISTICS lines create an entity that will provide unlimited logistic support to a **CombatUnit**. A name, troop strength, and location are specified. The WPN lines define weapon systems that will be added to the **FlagCombatUnits** when they are constructed. The WPN line specifies a name for the weapon, a firing rate (in units of issue per hour), kills per unit of issue fired, and the name of the **SupplyType** that the weapon consumes. These lines must appear prior to the lines that construct **FlagCombatUnits** but after the **SupplyTypes** are defined.

While **FlagCombatUnits** are **CombatUnits**, they cannot be constructed in the control file in the manner the **StressCombatUnits** were constructed. The **FlagCombatUnit** requires additional data. Therefore, they are constructed using the FLAGCOMBATUNIT key word. The FLAGCOMBATUNIT line specifies a name, a detection range, the side of the conflict, and the speed the unit moves. Immediately following the FLAGCOMBATUNIT line, the ADDWPN lines add a previously defined weapon system to the **FlagCombatUnit** just constructed.

The remaining difference is the use of the SUPPLIER lines to specify a set of supply rules more complex than the DEFAULTSUPPLIER line. The SUPPLIER lines specify whom to order from for a specific **SupplyType** when ordered at the specified priority. When multiple lines specify multiple suppliers for the same unit, **SupplyType**, and priority, the ordering unit will try the suppliers in the order the SUPPLIER lines appear in the file until the order can be filled.

These two examples were meant to give the reader a feel for how and in what order the key words must appear in the control file to build a model. For a complete explanation of the key words see Appendix C.

Once the control file is constructed, there are two methods for running the model. The model can be executed on a single computer or run in parallel on multiple platforms. Prior to running the model, FLEXLOGS must be correctly installed. To install FLEXLOGS, place the file "flexlogs.zip" into a directory. Then add flexlogs.zip to the CLASSPATH on the computer. For example, if flexlogs.zip is placed in the model directory on the C drive of a Windows computer, add "C:\model\flexlogs.zip" to the

CLASSPATH.[5] The installation of FLEXLOGS can be verified by typing "java Version" at a command prompt. This should print out version information for FLEXLOGS.

In order to run the model on a single computer, one needs only to type "java Model *infile outfile*" at a prompt, where *infile* is the name of the input control file. The results of each data point will be written to a comma delimited file specified by *outfile.*

To run the model on multiple computers in the Client/Server mode, one computer must be chosen as the server for the execution. On the server computer, type "java ModelServerMain *port infile outfile*". The *port* is the base IP port that the server will use. The server actually uses three ports (*port*+0, *port*+1, and *port*+2). The *infile* and *outfile* are the same as above. On each of the client machines, type "java ModelClient *address port* ". The *address* is the IP address of the computer running as the server. The *port* is the same port that was specified when the server was started. If the arguments to ModelClient are left out, *address* defaults to localhost (127.0.0.1) and *port* defaults to 7925. The client can be run on any number of computers that are connected to the server computer.[6] The results from the runs on the client computers are sent back to the server and logged in the file specified.

---

[5] The exact installation procedure and method of setting the CLASSPATH will vary from platform to platform so a comprehensive set of installation instructions is beyond the scope of this thesis.

[6] Note that some security policies will not allow connection to an arbitrary port. If the computers with this policy in effect are used, the Client/Server arrangement will not function correctly. Also on some computers, the server will not respond to a client running on the same computer.

APPENDIX D CONTROL FILE KEY WORD DEFINITIONS

ADDWPN/*weaponType*//

Adds a weapon system of the given type (created by WPN) to the last FlagCombatUnit constructed. Must come after the WPN line that creates the weapon type and after the FlagCombatUnit to which the weapon will be added.

COMBATUNIT/suborg//

Creates a CombatUnit of the class set by SETDEFAULTCOMBATUNIT. Must appear after the line that creates the logistics support and after the SETDEFAULTCOMBATUNIT line if used.

Suborg – the SubOrganization or other Logistics that supports the combatUnit.

CONSUME/*supplyType*/*time*//

Schedules a consume event for the last constructed CombatUnit. Must appear immediately after the line that creates the CombatUnit.

SupplyType – the type of supply to consume

Time – the time to schedule the event

DEBUG/*class*//

Sets debug for the specified class. Class must be in the form *package.class*. If the word ALL is used in place of a class, then debug is set for all classes (see SETDEBUG). The class must have a static setDebug(boolean) method.

class- the name of the class.

DEFAULTSUPPLIER/*customer/supplier//*

Determines who a SubOrganization should order supplies from in the absence of any other information. (see SUPPLIER) This line must appear after the SubOrganizations are created.

customer – the SubOrganization that will order supplies.

supplier – the SubOrganization that the customer should order supplies from if no supplier is specified by a SUPPLIER card.

DISTDIV/*divisor//*

Allows all of the random variables in the supply package to be divided by a number. This has the effect of reducing all of the mean delays in the logistics model. Can appear anywhere in the control file prior to the RUN line.

divisor – The amount to divide the random numbers by.

DUMP//

Dumps the current information about the model to a file called "maker.dump". Can appear anywhere in the control file before the RUN line, however the dump will only contain the data up to the point in the control file at which DUMP appears.

FLAGCOMBATUNIT/*logistics/maxBattleTime/side/speed//*

Creates a new FlagCombatUnit. Must appear after the line that creates the logistics (either SUBORG or ORANGELOGISTICS) and after the OBJECTIVE lines.

logistics- The source of logistics support.

maxBattleTime – The length of time after which the unit will retreat from a battle if it is still in progress.

side – The side of the conflict that this unit is on. (BLUE or ORANGE)

speed – The speed that this unit moves.

LINK/*node1/node2/mode/status/length//*

Creates a fixed length Link between the two nodes given. Must appear after the NODE lines.

node1, node2 – the index of the nodes.

mode – AIR, LAND, PIPE, RAIL, SEA, WATER

status – FAST, MED, SLOW, DESTROYED

length – The length of the link.

LOG/suborganization/attribute/file//

Opens a log file for the given SubOrganization's attribute. Must appear after the SubOrganization is created. Should not be used in a control file used for multiple runs, since the last run will overwrite the file.

suborganization – the SubOrganization whose attribute is to be logged.

attribute – the name of the attribute. (must be a non-array)

file – the file name to write to data to.

LOGPOSITION/logistics/file//

Logs the position of either a SubOrganization or a OrangeLogistics to the file. Must appear after the logistics is created. Should not be used in a control file used for multiple runs, since the last run will overwrite the file.

logistics – the name of the SubOrganization or OrangeLogistics

file – the name of the log file.

LOGSUPPLY/suborganization/attribute/supplyType/file//

Logs the attribute for the given SupplyType. Must appear after the SubOrganization is created and all of the SupplyTypes are defined. Should not be used in a control file used for multiple runs, since the last run will overwrite the file.

suborganization – the SubOrganization whose attribute is to be logged.

attribute – the name of the attribute (must be an array).

supplyType – the name of the SupplyType to log.

file – the file name to write to data to.

NODE/*x-coord/y-coord*//

Creates a new Node at the given coordinates. Note that node numbers are assigned in the order the nodes are created. Must appear after the SITE line.

NODES/number//

Creates the given number of nodes at the origin. Note that node numbers are assigned in the order the nodes are created. Must appear after the SITE line.

number – the number of Nodes to create.

OBJECTIVE/side/xCoord/yCoord//

Creates an Objective.

side – the side of the conflict that owns the Objective (BLUE or ORANGE)

xCoord, yCoord – the coordinates of the Objective.

ORANGELOGISTICS/name/strength/xCoord/yCoord//

Creates an OrangeLogistics. Must appear before the creation of any object that uses the OrangeLogsitics.

name – the name of the unit.

strength – the troop strength of the unit.

xCoord, yCoord – the initial coordinates of the unit.

REM/

Causes the rest of the line to be ignored. Note that if there are variables of the form #var# in the line, the driver will still attempt to complete the substitutions. To prevent this change the '#' to '*' in lines which have been commented out.

RUN/time//

Starts the simulation. Should be the last line in the control file.

time – The time to stop the simulation. If 0, no termination is scheduled and the simulation will run until there are no more events scheduled.

RUNS/number//

Sets the number of replications per data point.

SETDEBUG//

Turns on debug for all objects in the simulation. Warning: this can produce a large amount of output and considerably slow down the execution of the model. This line can appear anywhere in the file before the RUN line.

SETDEFAULTCOMBATUNIT/class//

Determines which class will be used to construct CombatUnits with the COMBATUNIT line. Must appear before any COMBATUNIT lines.

class – the name of the Class used to construct CombatUnits. The name must be fully qualified (i.e. supply.CombatUnitImpl) The class must have a constructor that takes a SubOrganization and a TimeMaster as its arguments.

SITE/*name//*

Creates the Site for the run. Should only appear once, before any Nodes are created.

SUBORGANIZATION/name/node/prioritySupplySet/criticalSupplySet/

supplyLevelSet/initialSupplySet/desiredSupplySet/eoqSet/strength//

Creates a SubOrganization. Must appear after the Node is created, all SupplyTypes are defined, and the needed supply sets are defined.

name – the name of the SubOrganization.

node – the location for the SubOrganization.

prioritySupplySet – the supply set below which the unit will order at high priority.

criticalSupplySet – the levels below which the SupplyTypes are considered critical

supplyLevelSet – The level below which the unit will not act as a supplier.

initialSupplySet – The initial stock levels.

desiredSupplySet – The desired stock levels for the unit.

eoqSet – The minimum order sizes.

strength – The troop strength of the unit.

SUBORG... same format as above, just less typing.

SUPPLIER/customer/supplier/supplyType/priority//

Adds an entry to the Logistics Knowledge Base. Must appear after the SubOrganizations are defined. These entries are searched in order until a supplier who has the required amount of supply is found. If none are found, the supplier from the default table will be used.

customer – the SubOrganization that will order supplies.

supplier – The SubOrganization to order the supplies from.

supplyType – The type of supply this entry corresponds to.

priority – P for high priority, N for non-priority.

SUPPLY/*name/unitOfIssue/density/cube/criticalityType*//

Creates a new SupplyType.

name – the name of the SupplyType.

unitOfIssue – the name of the unit of issue.

density – the weight per unit of issue.

cube – the volume per unit of issue.

criticalityType – COMBAT, MOBILITY, OTHER

SUPPLYSET/name/default/type/val/type/val/...../ENDSET//

Will create an array for use in constructing SubOrganizations. All SUPPLY lines must appear prior to this so the arrays will be the correct size.

name – the name of the set.

default – the quantity for any SupplyType not specifically listed.

type – name of a previously constructed SUPPLY.

val – the amount of that type in the set.

SUPPLYSETMULT/newName/oldName/mult//

Takes the values in SUPPLYSET oldName and multiplies them by mult and costructs a SUPPLYSET called newName. Useful for constructing sets based on various days of supply or scaling sets.

TRANSPORTATION/name/home/mode/cube/weight/numberOfConditions/speed /.../speed/range/.../range/number//

Constructs Transportation assets. Must appear after the SubOrganization that is its home is constructed.

home – Name of the SubOrganization that owns the Transporatation Asset.

mode – AIR, LAND, PIPE, RAIL, SEA, WATER.

cube – The net capacity in cubic.

weight – The net weight capacity.

numberOfConditions – The number of link conditions in the model (currently 3). 0 indicates speed and range are independent of link condition.

speed – The speed of the Transportation (link order is FAST, MED, SLOW).

range – The max range of the Transportation (currently not used).

number – the number if identical assets to create.

VARIBLELINK/node1/node2.mode/status//

Creates a variable length Link between the two nodes given. Must appear after the NODE lines.

node1, node2 – the index of the nodes.

mode – AIR, LAND, PIPE, RAIL, SEA, WATER

status – FAST, MED, SLOW, DESTROYED

WPN/name/firingRate/pk/supplyType//

Creates a new weapon type. Must appear after the SupplyTypes are defined but before any ADDWPN lines.

name – The name of the weapon system.

firingRate – The maximum firing rate in units of issue per hour.

pk- The number of kills per unit of issue fired (Can be > 1.0)

supplyType – The type of supply this weapon consumes.

APPENDIX E SAMPLE INPUT CONTROL FILES

Control File that uses StressCombatUnit

```
SITE/small war/
RUNS/5//
VAR/premium/0/1000/100//
VAR/dos/500/1600/100//
NODES/13/
LINK/0/1/SEA/FAST/4800/ conus to jpn
LINK/0/1/AIR/FAST/4800/
LINK/0/2/SEA/FAST/5600/ conus to pod
LINK/0/2/AIR/FAST/5600/
LINK/1/2/SEA/FAST/1000/ jpn to pod
LINK/1/2/AIR/FAST/1000/
LINK/2/3/LAND/FAST/500// pod to fs1
LINK/2/3/AIR/FAST/500// pod to fs1
LINK/2/4/LAND/FAST/500// pod to fs2
LINK/2/4/AIR/FAST/500// pod to fs2
LINK/3/5/LAND/FAST/200//
LINK/3/5/AIR/FAST/200//
LINK/3/6/LAND/FAST/200//
LINK/3/6/AIR/FAST/200//
LINK/3/7/LAND/FAST/200//
LINK/3/7/AIR/FAST/200//
LINK/3/8/LAND/FAST/200//
LINK/3/8/AIR/FAST/200//
LINK/4/09/LAND/FAST/200//
LINK/4/09/AIR/FAST/200//
LINK/4/10/LAND/FAST/200//
LINK/4/10/AIR/FAST/200//
LINK/4/11/LAND/FAST/200//
LINK/4/11/AIR/FAST/200//
LINK/4/12/LAND/FAST/200//
LINK/4/12/AIR/FAST/200//

SUPPLY/bullets/box/180/1/COMBAT/
SUPPLYSET/2000/2000/ENDSET//
SUPPLYSET/1dos/#dos#/ENDSET/
REM/SUPPLYSET/1sup/*sup*/ENDSET//
SUPPLYSETMULT/1sup/1dos/2.5// Seemed most efficient.
SUPPLYSETMULT/2sup/1sup/2//
SUPPLYSETMULT/3sup/1sup/3//
SUPPLYSETMULT/4sup/1sup/4//
SUPPLYSETMULT/8sup/1sup/8//
SUPPLYSETMULT/16sup/1sup/16//
SUPPLYSETMULT/20dos/1dos/20/
```

```
SUPPLYSETMULT/120dos/1dos/120/
SUPPLYSETMULT/2dos/1dos/2//
SUPPLYSETMULT/8dos/1dos/8//
SUPPLYSETMULT/16dos/1dos/16//
SUPPLYSET/huge/100000000/ENDSET/
SUPPLYSETMULT/3dos/1dos/3/
SUPPLYSETMULT/4dos/1dos/4/
SUPPLYSET/one/1.0/ENDSET/
SUPPLYSET/zero/0.0/ENDSET/
SUBORG/conus/0/zero/zero/zero/huge/zero/one/1.0/
SUBORG/jpn/1/zero/zero/zero/2sup/3sup/one/1.0//
SUBORG/pod/2/8sup/1sup/zero/16sup/16sup/one/1//
SUBORG/fs1/3/4sup/1sup/zero/8sup/8sup/one/1//
SUBORG/fs2/4/4sup/1sup/zero/8sup/8sup/one/1//
SUBORG/a1/5/2dos/2000/2dos/3dos/3dos/one/1//
SUBORG/a2/6/2dos/2000/2dos/3dos/3dos/one/1//
SUBORG/a3/7/2dos/2000/2dos/3dos/3dos/one/1//
SUBORG/a4/8/2dos/2000/2dos/3dos/3dos/one/1//
SUBORG/b1/9/2dos/2000/2dos/3dos/3dos/one/1//
SUBORG/b2/10/2dos/2000/2dos/3dos/3dos/one/1//
SUBORG/b3/11/2dos/2000/2dos/3dos/3dos/one/1//
SUBORG/b4/12/2dos/2000/2dos/3dos/3dos/one/1//
TRANS/TEU/conus/SEA/27000/700000/0/30/5000/500/
TRANS/PLANE/conus/AIR/2000/3200/0/500/5000/500/ prem
REM/TRANS/TEU/jpn/SEA/2700/70000/0/30/5000/500/
REM/TRANS/PLANE/jpn/AIR/2000/3200/0/500/5000/#premium#/
TRANS/BIGTRUCK/pod/LAND/3700/70000/0/10/5000/500/
TRANS/PLANE/pod/AIR/2000/3200/0/500/5000/#premium#/
TRANS/10 helo/fs1/AIR/2000/50000/0/300/500/#premium#/
TRANS/truck/fs1/LAND/3850/100000/0/10/500/500/
TRANS/10 helo/fs2/AIR/2000/50000/0/300/500/#premium#/
TRANS/truck/fs2/LAND/3850/100000/0/10/500/500/
DEFAULTSUPPLIER/pod/conus/
DEFAULTSUPPLIER/jpn/conus//
DEFAULTSUPPLIER/fs1/pod//
DEFAULTSUPPLIER/fs2/pod//
DEFAULTSUPPLIER/a1/fs1//
DEFAULTSUPPLIER/a2/fs1//
DEFAULTSUPPLIER/a3/fs1//
DEFAULTSUPPLIER/a4/fs1//
DEFAULTSUPPLIER/b1/fs2//
DEFAULTSUPPLIER/b2/fs2//
DEFAULTSUPPLIER/b3/fs2//
DEFAULTSUPPLIER/b4/fs2//
SETDEFAULTCOMBATUNIT/combat.StressCombatUnit//
COMBATUNIT/a1//
CONSUME/bullets/0//
```

```
COMBATUNIT/a2//
CONSUME/bullets/0//
COMBATUNIT/a3//
CONSUME/bullets/0//
COMBATUNIT/a4//
CONSUME/bullets/0//
COMBATUNIT/b1//
CONSUME/bullets/0//
COMBATUNIT/b2//
CONSUME/bullets/0//
COMBATUNIT/b3//
CONSUME/bullets/0//
COMBATUNIT/b4//
CONSUME/bullets/0//
DUMP
REM/SETDEBUG//
RUN/1000/
```

A control file that uses FlagCombatUnit

```
REM/
RUNS/30//
VAR/dos/200/500/100//
VAR/premium/0/50/10//
SITE/S E A//
NODES/3//
NODE/0/0// Pusan
NODE/0/0// Def1
NODE/0/0// Def2
NODE/0/0// Def3
NODE/20/250// Off1
NODE/40/250// Off2
NODE/-20/250// Off3
NODE/-40/250// Off4
LINK/0/1/SEA/FAST/2100// conus to HI
LINK/1/2/SEA/FAST/3400// HI to JPN
LINK/2/3/SEA/FAST/700// JPN to Pusan
LINK/2/3/AIR/FAST/600// JPN to Pusan
LINK/0/3/SEA/FAST/5600// conus to pusan
LINK/0/3/AIR/FAST/5600// conus to pusan
LINK/1/3/SEA/FAST/4200// HI to pusan
LINK/1/3/AIR/FAST/4200// HI to pusan
REM/ Links between Pusan and defensive units.
VARIABLELINK/3/4/AIR/FAST//
VARIABLELINK/3/4/LAND/FAST//
VARIABLELINK/3/5/AIR/FAST//
```

```
VARIABLELINK/3/5/LAND/FAST//
VARIABLELINK/3/6/AIR/FAST//
VARIABLELINK/3/6/LAND/FAST//
REM/ Links between defensive units.
VARIABLELINK/4/5/AIR/FAST//
VARIABLELINK/4/6/AIR/FAST//
VARIABLELINK/6/5/AIR/FAST//
REM/ Links between Pusan and Offensive units.
VARIABLELINK/3/7/AIR/FAST//
VARIABLELINK/3/7/LAND/FAST//
VARIABLELINK/3/8/AIR/FAST//
VARIABLELINK/3/8/LAND/FAST//
VARIABLELINK/3/9/AIR/FAST//
VARIABLELINK/3/9/LAND/FAST//
VARIABLELINK/3/10/AIR/FAST//
VARIABLELINK/3/10/LAND/FAST//
REM/ Links between Offensive Units.
VARIABLELINK/7/8/AIR/FAST//
VARIABLELINK/8/9/AIR/FAST//
VARIABLELINK/9/10/AIR/FAST//

SUPPLY/bullets/case/180/1/COMBAT//
SUPPLY/shells/ea/500/10/COMBAT//
SUPPLYSET/1dos/#dos#/ENDSET//
SUPPLYSET/1combat/400/ENDSET//
SUPPLYSETMULT/2combat/1combat/2//
SUPPLYSETMULT/3combat/1combat/3//
SUPPLYSET/zero/0/ENDSET//
SUPPLYSETMULT/3dos/1dos/3//
SUPPLYSETMULT/2dos/1dos/2//
SUPPLYSETMULT/10dos/1dos/10//
SUPPLYSETMULT/35dos/1dos/35//
SUPPLYSETMULT/50dos/1dos/50//
SUPPLYSETMULT/999dos/1dos/99999//

OBJECTIVE/BLUE/0/0//
OBJECTIVE/ORANGE/0/600//

ORANGELOGISTICS/orange defence/1400/0/600//
ORANGELOGISTICS/o1/1000/20/350//
ORANGELOGISTICS/o2/1000/40/350//
ORANGELOGISTICS/o3/1000/-20/350//
ORANGELOGISTICS/o4/1000/-40/350//

SUBORG/conus/0/zero/zero/zero/999dos/zero/1dos/1//
SUBORG/hi/1/3dos/1dos/zero/50dos/50dos/10dos/1//
SUBORG/jpn/2/3dos/1dos/zero/35dos/35dos/10dos/1//
```

```
SUBORG/pusan/3/3dos/1dos/zero/10dos/10dos/3dos/1//
SUBORG/def1/4/2dos/1dos/2dos/3dos/3dos/1dos/2000//
SUBORG/def2/5/1dos/1dos/2dos/3dos/3dos/1dos/2000//
SUBORG/def3/6/1dos/1dos/2dos/3dos/3dos/1dos/2000//
SUBORG/off1/7/1dos/1dos/2dos/3dos/3dos/1dos/2000//
SUBORG/off2/8/1dos/1dos/2dos/3dos/3dos/1dos/2000//
SUBORG/off3/9/1dos/1dos/2dos/3dos/3dos/1dos/2000//
SUBORG/off4/10/1dos/1dos/2dos/3dos/3dos/1dos/2000//

WPN/rifle/1/.02/bullets//
WPN/artillery/.1/.5/shells//

REM/ BLue units
FLAGCOMBATUNIT/def1/10/BLUE/1//
ADDWPN/rifle//
ADDWPN/artillery//
FLAGCOMBATUNIT/def2/10/BLUE/0.75//
ADDWPN/rifle//
ADDWPN/artillery//
FLAGCOMBATUNIT/def3/10/BLUE/0.5//
ADDWPN/rifle//
ADDWPN/artillery//
FLAGCOMBATUNIT/off1/10/BLUE/1//
ADDWPN/rifle//
ADDWPN/artillery//
FLAGCOMBATUNIT/off2/10/BLUE/1//
ADDWPN/rifle//
ADDWPN/artillery//
FLAGCOMBATUNIT/off4/10/BLUE/1//
ADDWPN/rifle//
ADDWPN/artillery//
FLAGCOMBATUNIT/off4/10/BLUE/1//
ADDWPN/rifle//
ADDWPN/artillery//
REM/ Orange units
FLAGCOMBATUNIT/orange defence/10/ORANGE/.5//
ADDWPN/rifle//
ADDWPN/artillery//
FLAGCOMBATUNIT/o1/10/ORANGE/1//
ADDWPN/rifle//
ADDWPN/artillery//
FLAGCOMBATUNIT/o2/10/ORANGE/1//
ADDWPN/rifle//
ADDWPN/artillery//
FLAGCOMBATUNIT/o3/10/ORANGE/1//
ADDWPN/rifle//
ADDWPN/artillery//
```

```
FLAGCOMBATUNIT/o4/10/ORANGE/1//
ADDWPN/rifle//
ADDWPN/artillery//

REM/ transportation assets
TRANS/TEU/conus/SEA/2700/70000/0/20/9999/500//
TRANS/TEU/hi/SEA/2700/70000/0/20/9999/500//
TRANS/TEU/jpn/SEA/2700/70000/0/20/9999/500//
TRANS/plane/conus/AIR/2000/3200/0/400/9999/#premium#//
TRANS/plane/hi/AIR/2000/3200/0/400/9999/#premium#//
TRANS/plane/jpn/AIR/2000/3200/0/400/9999/#premium#//
TRANS/5T/pusan/LAND/385/10000/0/30/9999/100//
TRANS/helo/pusan/AIR/200/5000/0/200/999/#premium#//
TRANS/helo/def1/AIR/200/5000/0/200/9999/1//
TRANS/5T/def1/LAND/385/10000/0/30/9999/10//
TRANS/helo/def2/AIR/200/5000/0/200/9999/1//
TRANS/5T/def2/LAND/385/10000/0/30/9999/10//
TRANS/helo/def3/AIR/200/5000/0/200/9999/1//
TRANS/5T/def3/LAND/385/10000/0/30/9999/10//
TRANS/helo/off1/AIR/200/5000/0/200/9999/1//
TRANS/5T/off1/LAND/385/10000/0/30/9999/10//
TRANS/helo/off2/AIR/200/5000/0/200/9999/1//
TRANS/5T/off2/LAND/385/10000/0/30/9999/10//
TRANS/helo/off3/AIR/200/5000/0/200/9999/1//
TRANS/5T/off3/LAND/385/10000/0/30/9999/10//
TRANS/helo/off4/AIR/200/5000/0/200/9999/1//
TRANS/5T/off4/LAND/385/10000/0/30/9999/10//
DEFAULTSUPPLIER/hi/conus//
DEFAULTSUPPLIER/jpn/hi//
DEFAULTSUPPLIER/pusan/jpn//
DEFAULTSUPPLIER/def1/pusan//
DEFAULTSUPPLIER/def2/pusan//
DEFAULTSUPPLIER/def3/pusan//
DEFAULTSUPPLIER/off1/pusan//
DEFAULTSUPPLIER/off2/pusan//
DEFAULTSUPPLIER/off3/pusan//
DEFAULTSUPPLIER/off4/pusan//

SUPPLIER/pusan/jpn/bullets/P//
SUPPLIER/pusan/jpn/shells/P//
SUPPLIER/pusan/hi/bullets/P//
SUPPLIER/pusan/hi/shells/P//
SUPPLIER/pusan/conus/bullets/P//
SUPPLIER/pusan/conus/shells/P//


REM/ allow canniblization by un remming
```

```
REM/SUPPLIER/def1/pusan/bullets/P//
REM/SUPPLIER/def2/pusan/bullets/P//
REM/SUPPLIER/def3/pusan/bullets/P//

REM/SUPPLIER/off1/pusan/bullets/P//
REM/SUPPLIER/off2/pusan/bullets/P//
REM/SUPPLIER/off3/pusan/bullets/P//
REM/SUPPLIER/off4/pusan/bullets/P//

REM/SUPPLIER/def1/def2/bullets/P//
REM/SUPPLIER/def2/def1/bullets/P//
REM/SUPPLIER/def2/def3/bullets/P//
REM/SUPPLIER/def3/def2/bullets/P//
REM/SUPPLIER/def1/def3/bullets/P//
REM/SUPPLIER/def3/def1/bullets/P//

REM/SUPPLIER/off1/off2/bullets/P//
REM/SUPPLIER/off2/off3/bullets/P//
REM/SUPPLIER/off3/off2/bullets/P//
REM/SUPPLIER/off4/off3/bullets/P//
REM/SUPPLIER/off2/off1/bullets/P//
REM/SUPPLIER/off3/off4/bullets/P//

REM/SUPPLIER/def1/pusan/bullets/P//
REM/SUPPLIER/def2/pusan/bullets/P//
REM/SUPPLIER/def3/pusan/bullets/P//

REM/SUPPLIER/off1/pusan/shells/P//
REM/SUPPLIER/off2/pusan/shells/P//
REM/SUPPLIER/off3/pusan/shells/P//
REM/SUPPLIER/off4/pusan/shells/P//

REM/SUPPLIER/def1/def2/shells/P//
REM/SUPPLIER/def2/def1/shells/P//
REM/SUPPLIER/def2/def3/shells/P//
REM/SUPPLIER/def3/def2/shells/P//
REM/SUPPLIER/def1/def3/shells/P//
REM/SUPPLIER/def3/def1/shells/P//

REM/SUPPLIER/off1/off2/shells/P//
REM/SUPPLIER/off2/off3/shells/P//
REM/SUPPLIER/off3/off2/shells/P//
REM/SUPPLIER/off4/off3/shells/P//
REM/SUPPLIER/off2/off1/shells/P//
REM/SUPPLIER/off3/off4/shells/P//

REM/ logs
```

```
REM/LOGSUPPLY/off1/curSupply/bullets/off1_bullets.txt//
REM/LOG/def1/strength/def1.txt//
REM/LOG/def2/strength/def2.txt//
REM/LOG/def3/strength/def3.txt//
REM/LOG/off1/strength/off1.txt//
REM/LOG/off2/strength/off2.txt//
REM/LOG/off3/strength/off3.txt//
REM/LOG/off4/strength/off4.txt//

REM/DEBUG/supply.Asset//
REM/DEBUG/TM.TimeMaster//
REM/DEBUG/battle.Lanchester//
REM/DEBUG/combat.FlagCombatUnit//
RUN/1999//
```

APPENDIX F SOFTWARE AND HARDWARE USED

The FLEXLOGS model was developed using Sun's Java Development kit version 1.1.6 and a text editor VIM ("vi improved") Version 5.1 [Bram Moolenaar et al., 1998].

The model was primarily run using Sun's Java Virtual Machine version 1.1.6. (Note that 1.1.6 is significantly faster than previous versions.)

The data analysis was conducted using S-plus version 4.5 from Mathsoft. (Note: version 4.5 is the first version to support the color coded 3-D graphs used in this research.)

This document was produced using Microsoft Word-97 and Powerpoint-97.

FLEXLOGS was developed primarily on a Gateway-2000 with a Pentium-90 processor and 80MB of RAM and an IBM ThinkPad 760ED (laptop) with a Pentium-133 processor and 16MB of RAM.

The model runs were conducted on a wide range of platforms including: The two systems mentioned above, Sun Sparc 20's, and a number of Dell Pentium-400 computers.

APPENDIX G AREAS FOR POTENTIAL IMPROVEMENT OF FLEXLOGS

During development of the model, a number of potential improvements of the model were identified. This appendix provides an explanation of some these improvements. No attempt was made to prioritize this list. Some of the changes required are minor, while others may result in a major change to the software. Some of the required changes are marked in the source code with the word "TODO" to allow for easy identification.

In battle.Referee, add a random element to the computeDetection method. Currently the range at which units detect is deterministic. This only affects the simulation when FlagCombatUnit is used.

In combat.ConsumptionEvent, implement the rate-based consumption event. Currently, only bulk consumption at a specific time is implemented. Implementing rate-based consumption will allow for more accurate modeling of the consumption of supplies during a battle.

In combat.FlagCombatUnit, make defensiveFactor, SensorRange, and StealthFactor able to be changed. These values currently take on default values that cannot be changed.

In combat.FlagCombatUnit, add some random elements to the outcome of a fight. The results of the Lanchester calculations are currently purely deterministic.

In combat.FlagCombatUnit, improve the fuel consumption model. The fuel consumption model is currently very simplistic. The unit consumes one unit of fuel per one unit of distance per person assigned to the unit.

In maker.CombatUnitMaker, add a check to make sure that the class provided does in fact implement the CombatUnit interface. The CombatUnitMaker assumes that the class set in the ModelMaker implements the interface without checking.

In maker.ModelMaker, add flags to improve the ability to check that the key words appear in the correct order.

In supply.Link, if a Link is destroyed, need to figure out what to do with any Transportation on the Link.

In supply.ModePool, the variable "util" that controls the minimum utilization for Transportation should probably be somewhere else and should be changeable at run time.

In supply.SubOrganization and Transportation, modify so that multiple Requisitions can share the same Transportation.

The ability to model pipelines should be added. The pipeline would require the modeling of a continuous flow Transportation.

Currently there is no use of fuel by Transportation assets modeled. This could be added by requiring the supplier to expend fuel for each Requisition shipped, based on the Transportation used and the distance shipped. The SubOrganization would also need a queue to hold Requisitions waiting for fuel.

One area of study not currently supported is the susceptibility of large stockpiles to attack. Modeling this would require adding an interdiction model to the combat model.

The random variate distributions should be able to be set at run time. This would require modifications to the ModelMaker to add the additional keywords and significant modifications to the class holding the distributions (Dist).

Add the ability in FlagCombatUnit for a unit to be destroyed. Currently a unit will continue to fight with smaller and smaller troop strengths and will not reach some value at which the unit must disband.

Add the ability to treat troops as a SupplyType so that they can be replenished. The key to implementation is some method of distinguishing them as a special SupplyType so that the SubOrganization and any CombatUnit can track them as the troop strength.

In the ModelDriver, a way to have a variable depend on the value of other variables should be developed to allow more flexibility in varying model parameters. The same change would also need to be made in the VariableServer class.

Modify the simulation to make it High Level Architecture (HLA) compliant. The logistics model and the combat model should be split and implemented separately. This is probably a thesis size project in itself, but would allow FLEXLOGS to be federated with an existing, validated combat model.

Incorporate a way to model the effects of information technology. This first requires a determination of what "information technology" means in this context.

REFERENCES

Atchesson, Michael J., "Cost-Benefit Analysis of the Enhanced Transportation Service Program," Master's Thesis, Naval Postgraduate School, Monterey, California, June 1997.

Bankes, Steven C., "Exploratory Modeling and the Use of Simulation for Policy Analysis," RAND, Santa Monica, California, 1992.

Brooks, A., Bankes, S., Bennett, B., "Weapon Mix and Exploratory Analysis, A Case Study," RAND, Santa Monica, California, 1997.

Buss, Arnold, OA-4333 class notes, Naval Postgraduate School, Monterey, California, 1998.

Buss, Arnold, OA-3302 class notes, Naval Postgraduate School, Monterey, California, 1997.

Buss, Arnold, OA-3200 class notes, Naval Postgraduate School, Monterey, California, 1997.

Halvorson, T. G., "Improvement to a Surge and Sustainment Model for Wargaming," Master's Thesis, Naval Postgraduate School, Monterey, California, 1994.

Huffaker, Joseph W., "Modeling the Effects of Logistics on Ground Combat and Maneuver," Master's Thesis, Naval Postgraduate School, Monterey, California, September 1996.

Joint Vision 2010, Joints Chiefs of Staff, Washington D.C.

Long, John A., "Modeling Theater Level Logistics for Wargames," Master's Thesis, Naval Postgraduate School, Monterey, California, December 1993.

Moolewaar, Bram et. al, "The ViM editor," Computer Software, 1998.

Noel, Jack S. II, "An Object-Oriented Logistics Over the Shore Simulation: An Aid in Throughput Estimation," Master's Thesis, Naval Postgraduate School, Monterey, California, September 1993.

Owens, William A., "The Emerging U.S. System of Systems." In Johnson, Stuart, and Libicki, Martin (Eds.) *Dominant Battlespace Knowledge*, National Defense University, Washington, D.C., 1995.

Plunk, Curtis D., "A Computer Simulation of Logistics Networks for Wargame Umpires," Master's Thesis, Naval Postgraduate School, Monterey, California, March 1995.

Seager, Kevin, personal conversation, June 1998.

Simulation Analysis Center (SAC) briefings, December 1997.

Simulation Analysis Center (SAC), "A Conceptual Object Model (COM) of Military Logistics (Draft)," December 1997.

Stuart, Chris, "Supporting Operational Maneuver from the Sea from a Sea Base", Master's Thesis, Naval Postgraduate School, Monterey, California, September 1998.

Stork, Kirk A., "Sensors in Object Oriented Discrete Event Simulation," Master's Thesis, Naval Postgraduate School, Monterey, California, September 1996.

Taylor, James G., *Lanchester Models of Warfare,* Operations Research Society of America, Arlinton, Va., March 1983.

Zhong, Caoyuan, "Modeling of Airport Operations Using an Object-Oriented Approach," PhD dissertation, Virginia Polytechnic Institute and State University, February 1997.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   8725 John J. Kingman Rd., STE 0944
   Ft. Belvoir, Virginia 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, California 93943-5101

3. Defense Logistics Studies Information Exchange 1
   U.S. Army Logistics Management College
   Fort Lee, Virginia 23801-6043

4. A. H. Buss, Code OR/Bu 1
   Department of Operations Research
   Naval Postgraduate School
   Monterey, California 93943-5000

5. Wayne Hughes, Code OR/Hl 1
   Department of Operations Research
   Naval Postgraduate School
   Monterey, California 93943-5000

6. Office of the Secretary of Defense 1
   Program Analysis and Evaluation
   Attn: LtCol Charles Mugno, USMC
   Pentagon, Room 2E314
   Washington, D.C. 20301-1800

7. Office of the Secretary of Defense 1
   Program Analysis and Evaluation
   Attn: Dr Kevin Seager
   Pentagon, Room 2E314
   Washington, D.C. 20301-1800

8. Office of the Secretary of Defense 1
   Program Analysis and Evaluation
   Attn: Mr. James Johnson
   Pentagon, Room 2E314
   Washington, D.C. 20301-1800

9. LCDR John Ruck 1
5641 Pembrook Dr.
New Orleans, Louisiana 70131

10. Chief of Naval Operations (N-81) 1
ATTN: Dr. Susan Marquis
Navy Department
Washington, DC 20350

11. Deputy Chief of Naval Operations (Logistics) 1
ATTN: LCDR Carolyn Kresek, N421C
2000 Navy Pentagon
Washington, DC 20350-2000

12. Chief of Naval Operations (N-816) 1
ATTN: Mr. Bruce Powers
Navy Department
Washington, D.C. 20350

13. D. A. Schrady, Code OR/Sd 1
Department of Operations Research
Naval Postgraduate School
Monterey, California 93943-5000